NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE NPS LOCATOR SYSTEM

by

Jeffrey E. Forte

December, 1997

Thesis Advisors: James C. Emery
C. Thomas Wu

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December, 1997	Master's Thesis

4. TITLE AND SUBTITLE
THE NPS LOCATOR SYSTEM

5. FUNDING NUMBERS

6. AUTHOR(S)
Forte, Jeffrey E.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The purpose of this thesis is to design, develop and implement a personnel locator system at the Naval Postgraduate School (NPS). A prototype locator system was developed and implemented on the NPS TCP/IP network. The locator provides information such as email addresses, phone and fax numbers, and building and office locations, as well as facilities such as hotlinks for email applications and homepages. In addition, the NPS Locator automatically updates its personnel information on a configurable time schedule. This thesis includes a discussion of the prototype development to include requirements, tools, and design. Some program code is included as appendices. This paper also discusses the benefits and considerations of intranet technology, and explores a popular Web application architecture on which the NPS Locator is based. Finally, this thesis makes recommendations for improvements to the NPS computing environment to allow for future intranet development.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Intranet, Directory, TCP/IP Networking, Web Application, HTTP, CGI	77
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFI- CATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

THE NPS LOCATOR SYSTEM

Jeffrey E. Forte
Captain, United States Marine Corps
B.S., United States Naval Academy, 1988

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN COMPUTER SCIENCE
and
MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
December 1997

Author: _____
Jeffrey E. Forte

Approved by: _____
James C. Emery, Thesis Advisor

_____
C. Thomas Wu, Thesis Advisor

_____
Dan Boger, Chairman
Department of Computer Science

_____
Reuben Harris, Chairman
Department of Systems Management

ABSTRACT

The purpose of this thesis is to design, develop and implement a personnel locator system at the Naval Postgraduate School (NPS). A prototype locator system was developed and implemented on the NPS TCP/IP network. The locator provides information such as email addresses, phone and fax numbers, and building and office locations, as well as facilities such as hotlinks for email applications and homepages. In addition, the NPS Locator automatically updates its personnel information on a configurable time schedule. This thesis includes a discussion of the prototype development to include requirements, tools, and design. Some program code is included as appendices. This paper also discusses the benefits and considerations of intranet technology, and explores a popular Web application architecture on which the NPS Locator is based. Finally, this thesis makes recommendations for improvements to the NPS computing environment to allow for future intranet development.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

The purpose of this thesis is to design, develop and implement a personnel locator system at the Naval Postgraduate School (NPS). The locator will provide information such as email addresses, phone and fax numbers, and building and office locations, as well as facilities such as hotlinks for email applications and homepages. The impetus for the NPS Locator system was twofold. The first spawned from the "Strategic Plan for Computing at the Naval Postgraduate School," published late 1995. In an effort to improve the computing infrastructure at NPS, the Associate Provost of Computing was interested in testing the viability of intranet technology as an application deployment platform for campus-wide systems. A locator system was selected as one of the first applications because of its relative simplicity, and because its usefulness should attract users, resulting in exposure for intranet technology at NPS. The second impetus arose from, simply, the lack of any existing accurate and complete directory system at the time this study was conceived.

A. THE NPS COMPUTING ENVIRONMENT

The Strategic Plan mentioned above addresses many different facets of the NPS computing environment. Two issues that are relevant to this thesis are the communications network and administrative application development. The network, as the document states, "is fragmented and unreliable, held together by bailing wire and sweat." The Associate Provost's goal is to have a network that is, "ubiquitous, reliable, transparent, and (that is) easy to use as a modern telephone system." The network is fragmented because, in the past, the enterprise perspective of the network was not acknowledged. Networks were built and administered independent of each other, usually along departmental lines. This independent growth resulted in a network that consists of at least four different network operating systems (NOS), various resource sharing and access capabilities, minimal interoperability, various email naming schemes, and different directory systems for each NOS.

The development of administrative applications also fell victim to this decentralized and haphazard approach to information technology management. Applications were developed and data sources created within an organizational unit without consideration of needs outside of the particular organizational unit. This lead to duplication of effort and did not foster interoperability. These "stovepipe" applications have different data requirements, different hardware requirements, and, for the most part,

are not widely accessible from outside the organizational unit in which they were developed.

These issues provoked a grass-roots exploration into intranet technology from several faculty, staff, and students including the Associate Provost of Computing. Several independent theses have been written to address various aspects of intranet technology and how it can be applied in the NPS computing environment. Chapter II discusses the benefits and considerations of intranets. Again, the focus of this work is to build an intranet application, the NPS Locator directory system, using industrial strength tools that are already being used by the NPS computing staff to serve as a demonstration of intranet viability at NPS.

B. SURVEY OF NPS DIRECTORIES

At the time this study was undertaken the directories that existed at NPS were inaccurate and incomplete. Accuracy in this context refers not only to phone numbers, email addresses, and descriptive information being outdated and incorrect, but also to misspellings of the data. Completeness of the directory refers to the content of the directories. For example, one directory, the phone book, only provides phone numbers, whereas the UNIX "finger" application may only provide email information. A complete directory should provide information to communicate or "locate" a person by all available means.

The preexisting directories included the NPS paper phone book, UNIX "ph" and "finger" programs, proprietary network directories (e.g., Banyan Vines StreetTalk Directory) and a variety of uncoordinated and unmanaged Web directories, some containing departmental information only. These directories also required frequent manual updates if they were to stay current.

These preexisting directories varied greatly in many aspects. By reviewing the directories, and correlating notes on their strengths and weaknesses, seven characteristics of directory systems became apparent. The characteristics later drove the requirements and design of the NPS Locator Prototype. The characteristics are as follows:

1. Scope

The scope of the directory refers to the type entities listed. For example, many of the directories only have staff and faculty in them. Others only have staff and faculty from within a particular department. The desired scope includes faculty, staff and students from the entire Naval Postgraduate School population.

2. Completeness

The completeness of the directory refers to the attributes listed on each entity in the directory. For example, some directories only list phone number, office and department. The desired list of entity attributes were chosen by testing each of them against the following premise: An attribute must enhance or enable either the process of locating an individual or the actual communication with the individual. For example, if an individual needs to communicate with another, the office number may enable the process by letting the first individual physically locate the second individual. A photograph may also aid in the process if the individuals have never met in person previously.

3. Accessibility

Accessibility is concerned, in part, with who has authorization to use the directory. Many of the directories were limited to NPS personnel use, others were available to the Internet community. In addition, accessibility deals with the ease of access to the directory. Some directories were deeply hidden in a department's own Web pages, and not publicly known or advertised to the rest of NPS. The directory should be available for all to use, with perhaps, various views depending on whether access is from outside or from within the NPS intranet. Furthermore, the directory should be easy to get to and should be made known to all of NPS. The most obvious location for an NPS-wide directory would be on the NPS homepage.

4. Platform Dependence

The directory should be available on a platform that is readily available to the largest population at NPS. This requirement unequivocally points to the Web as the platform of choice, since most NPS personnel have access to the Internet and the Web. Some of the directories reviewed either required the user to be on a proprietary LAN or required the user to use a particular operating system. For example, users can only access the StreetTalk Directory System from a Banyan Vines network client.

5. Updates

All of the directories surveyed required manual updating. Few, if any, had any regular update schedule. It is important to note, however, that most of the directories only contained staff and faculty listings, which contain rather static attributes. The

student population changes every quarter, and therefore provides more of a challenge to the directory update issue. It is preferable to automate the update process for two reasons. Firstly, updating the directory would most probably be a collateral duty for the responsible person, and certainly would not receive first priority. Maintaining a directory manually can be a tedious undertaking, especially if the directory is robust with information from a variety of data sources. With an automated update process, increasing the frequency of updates adds no burden to an individual, and because it is automated, it can be run at off-peak hours, therefore not impacting on NPS computer resources during peak hours.

6. Data Sources

The surveyed directories used a variety of data sources. Some directories received their data from the data owners initially, but then did not refresh that data on a regular basis, if at all. Others used second-hand data sources, such as departmental databases, which probably originated from the data owners. Again, no update method was in place to synchronize the data with the data owner's data source. Still others created their data locally.

The desired data sources are those that reside with the data owners. The data owners should have the most accurate and update information on the entities that they are responsible for. For example, the Human Resources Office (HRO) should have the most accurate information for civilian staff and faculty. This is not necessarily true for all attributes. For example, civilian faculty homepage addresses and class schedules would not be under the HRO ownership. The Registrar would own the class schedules and perhaps the computing center would own the homepage information. The data ownership issue assumes that the owner is properly maintaining the data.

7. User Interface

Finally, the user interfaces of the surveyed directories varied greatly. Some were command-line oriented, such as the "ph" interface, which is clumsy and not very powerful. Others varied between DOS menu systems and Windows point-and-click interfaces with varying degrees of complexity and usefulness. A few Web-based directories had good interfaces that were indeed easy to use.

A directory should be easy to use and understand. The Web browser, with its point-and-click nature, provides users with an intuitive way to navigate through applications and locate the information they need using a common look and feel. The

directory's interface design within the browser still requires thought, however, to ensure it is predictable and intuitive.

C. THESIS ORGANIZATION

This thesis has two main logical parts. The first part looks at intranets, providing a fairly thorough introduction into the technology and into the benefits and considerations of implementing an intranet. In addition, a generic web application architecture is discussed to provide some background into the architecture of the NPS Locator System.

The second logical part explains the development of the NPS Locator from requirements through the functional capabilities of the prototype implementation. The outline of this thesis is, therefore, as follows:

- Chapter I. Introduction. This chapter provides an overview of the paper and discusses preexisting directories at NPS.
- Chapter II. Introduction to Intranets. This chapter introduces intranet technology, its benefits and considerations, and ends with a discussion of a common web application architecture.
- Chapter III. NPS Locator Development. Chapter III provides the central focus of this thesis, the development of the NPS Locator System. This chapter reviews the requirements, tools, architecture, and functional aspects of the Locator.
- Chapter IV. Conclusion and Recommendations.

II. INTRODUCTION TO INTRANETS

The NPS Associate Provost of Computing is not alone with his interest in intranet technology. Numerous corporations and government installations are aggressively building intranets within their organizations. This interest comes on the heels of the recent success of the Internet and the World Wide Web (WWW). Many organizations want to exploit this inexpensive Web technology (Web browsers and Web servers). This section will define intranet, as well as discuss the technology and the benefits that this technology provides.

A. INTRANET DEFINED

An intranet is an internal network based on the same technologies used on the Internet and the World Wide Web. The Internet is the global internetwork (i.e., network of networks) that links tens of thousands of organizations—government, commercial and educational—from all over the globe. The Internet began as a Department of Defense (DOD) project in the late 1960's. The growing importance of computers at that time gave rise to multiple challenges both in sharing information among different locations with different hardware and with keeping information intact during potential network disruptions at an individual site. The project resulted in the building of a network that interconnected various educational institutions and government installations. This network was called ARPANET (Advanced Research Projects Agency Network), which evolved into what now is known as the Internet.

The Internet utilizes the TCP/IP (Transmission Control Protocol/Internet Protocol) suite to interconnect these various computers and networks. IP allows these distributed networks to route and pass information to each other independently, so if one site is down the information can be routed through alternate sites to its final destination. The wide use of IP protocol, and hence the Internet, has made IP a de facto standard. (Telleen, 1996)

Web technology has its roots in research carried out at the European Particle Physics Laboratory, CERN, in the early 1990's. This project focused on a way of sharing information using hypertext links. Hypertext links are special codes that are embedded in a document that provide a way to jump to another part of the same document or to a different document. Initially, the capability was limited to linking documents within a single computer. The functionality was extended to allow one to select documents (or pages) on other computers at different sites on the Internet. The hypertext transfer

protocol (HTTP) was the protocol developed to provide this "remote" capability. HyperText Markup Language (HTML) is the file standard used in developing the hypertext documents or pages. It defines the "codes" necessary to create hypertext links as well as codes to enhance the presentation of a page. HTML documents can be very dynamic, often containing audio, video, and animation.

Web technology has two main components: a server (HTTP server) and a client (or web browser). The HTTP server (more often referred to as a Web server) serves or transfers an HTML document to the web browser. The web browser interprets the HTML document and displays it to the screen. The process is as follows:

- Your browser reads a document written in HTML and displays it for you, interpreting all the markup codes in the document.
- When you click on a hypertext link (hyperlink) in that document, your browser uses HTTP to send a network request to a Web server to access the new document or service specified by the hyperlink.
- Also using the HTTP protocol, the Web server responds to the request with the document or other data your requested.
- Your browser software then reads and interprets that information and presents it to you in the correct format. (Evans, 1996)

In summary, an intranet is a TCP/IP network which is internal to your organization, containing HTTP servers that provide HTML documents and other services to Web browsers (e.g., Netscape's Navigator or Microsoft's Internet Explorer) running on various hardware platforms throughout the network.

B. INTRANET BENEFITS

This section outlines the more prevalent benefits that result from instituting an intranet. This list is not confined to NPS, but focuses on how organizations in general can benefit from this technology. Other benefits may be realized because of the particular nature of an organization's business, information, and communication requirements; the size of the organization; and the geographical locations of its components. These various aspects may align well with the intranet concepts and then allow for greater benefits.

1. Low Implementation Costs

Web technology is based on mature and "truly open standards such as TCP/IP, HTTP, HTML, CGI, and Java, and thus does not lock organizations into proprietary technology with limitations and significant costs." (Linthicum, DBMS 1997) The hardware and software components necessary to get an intranet up and running are cheap. Often, most of these components are already in place. For example, most organizations have some sort of proprietary network already running, and adding TCP/IP to these networks is usually a simple task. Moreover, most network operating systems today are shipped with their own TCP/IP stack, eliminating some of the cost for an intranet. In addition, many PCs today come with a Web browser already loaded. Even if these components had to be purchased, they are relatively inexpensive.

Furthermore, "Web applications driven from a single browser require less training, and deployment costs are reduced due to the single interface, protocol, and middleware architecture. Large organizations report costs of less than $40 per user to create and intranet infrastructure, compared to approximately $2000 per user for traditional two-tier client-server application infrastructure." (Linthicum, DBMS 1997)

2. Universal Client

Another benefit of intranets is the multi-platform nature of Web browsers. Browsers exist for most operating systems that are windows-based or that provide some sort of graphical user interface (GUI). These include UNIX (X-Windows), Microsoft Windows (3.11, 95, and NT), Macintosh, and OS/2. The availability of browsers across these diverse platforms is significant. The operating environments of the author, Web and applications servers, and the client viewer are independent of each other. Any document adhering to HTML and the related standards which is served by an HTTP server can be accessed by any client regardless of the operating environment from which they originated. (Telleen, 1996)

The ubiquity of the browser, therefore, brings two obvious benefits. The first is that the user only needs to learn one interface, allowing the user the opportunity to become proficient with the one interface. Also, this allows training and support costs to be focused in one direction, and should result in a lower costs overall.

The second benefit deals with classic client-server application deployment issues. One of the most costly aspects of client-server development in a multiplatform environment is developing and maintaining a separate client interface for each platform. This benefit of Web browsers is important to the concept of "thin-client" architecture. In

intranet-based applications, the application logic is located either on a Web server or an application server. Therefore, changes to the application can be made centrally, without the need to update and re-deploy the client, which would require considerable time and cost in a mulitplatform environment.

3. Simple and Intuitive Interface

The Web browsers use of point-and-click hyperlinking technology allows users to navigate and locate information easily. This type of interface is very intuitive and easy to learn. Furthermore, since the users are using one interface to access information and applications, they can quickly become proficient with the Web pages and Web applications.

4. Information Access

The Web information publishing paradigm has two primary advantages. The first advantage deals with the "push-vs.-pull" information distribution debate. In the "push" mode, "all" information is sent to the user, and the user must decide whether to read or discard the information. In the "pull" mode, users only pull what they need. The "pull" mode allows a person to be more efficient and selective in what they process, instead of wasting their time sifting through hoards of mostly non-essential information. The Web is well-suited for this. Information providers can publish the information and place it on a Web server. Users that want to access the information, have the choice to "pull" it down if they decide it is essential to their needs, or choose to ignore it if it is not essential to their needs.

The second advantage rises from the simplicity in updating Web content. For example, in the past, organizations published employee handbooks once a year and distributed them to all employees. Traditionally, even though these handbooks may have been out of date the day they were distributed, they would not be updated until the following year. With an intranet publishing strategy, information can be updated instantly, and there is no need for reprinting, duplicating, and distributing reams of ink and paper. Duplication and distribution are achieved by simply placing the handbook on a Web server. In support of this intranet benefit, "a recent International Data Corp. (IDC) study pegged ROI (return on investment) at 1,000 percent or better—mostly in paper and labor costs." (Currid, 1997)

5. Multimedia Content

Besides the quick and easy access to timely information, the information itself can be in a variety of formats, including text, graphics, video, audio, and animation. This can enhance the exchange of information and allow for different types of information to be conveyed. Web browsers already provide capabilities for these various formats, and therefore intranets provide this benefit at no extra cost.

C. INTRANET CONSIDERATIONS

Intranets are not the panacea for all computing and information needs of an organization. The benefits discussed above do not come free without certain considerations. Three main considerations are explored below:

1. Ease of Use

Though "ease of use" is a benefit of Web publishing on intranets, it also can become a negative attribute. For example, it is easy for any person in an organization to start putting content out on the internal Web. If not managed in some way, issues of information glut, security, privacy, and performance may become significant, resulting in more problems and a negative ROI. Therefore, much consideration into how the information infrastructure is to be managed must be put forth before the intranet takes off. How to manage the content producing infrastructure in an intranet is a topic of great debate and is beyond the scope of this thesis. In short, the debate centers around centralized or decentralized control of content production. If control is decentralized, how does one ensure the content developed adds value to the organization, or at a minimum, does not detract from the mission? On the other hand, if control is centralized, is the effectiveness and creativity of the organization greatly limited? Most successful intranets seem to find a balance between the two approaches to content management.

2. Security

The "ease of use" issue also highlights the need for security. If information is easily accessible, then extra care must be taken that only authorized users are allowed access to certain pieces of information. Most organizations have Internet access, and therefore a potential hole into the intranet exists via this Internet connection. Firewalls are special workstations that control access between your intranet and the Internet. Great

care must be taken in the administration of firewalls and other security measures because of the speed with which information can flow through an intranet and out to the Internet. For example, if an administrator incorrectly sets permissions on a document outlying future strategies of a corporation that only the executive staff is supposed to see, and the whole Internet has access to it, competitors may view it and restructure their business accordingly.

3. Interactive Application Development

The traditional Web technology was never designed to support interactive applications. HTTP and HTML form a "request-download-view-disconnect" type of architecture and cannot process applications interactively. (Linthicum, DBMS 1997)

Interactive applications require persistent connections between client and server. HTTP makes a connection, downloads a document, and then closes the connection. It has no mechanism to maintain a persistent connection. Furthermore, the original Web architecture could only deliver static content. Web pages had to be prepared beforehand, and placed on the web server for delivery. No mechanism existed to deliver dynamic content. In other words, there was no way to create pages on-the-fly. This capability is necessary for interactive application development.

D. WEB APPLICATION ARCHITECTURE

Originally, as stated above, the Web could only publish or present static content. With the introduction of CGI (common gateway interface), a Web server could then deliver dynamic content back to the user's Web browser. CGI is a protocol that defines how an external process on a Web server can create dynamic Web pages. CGI programs work in the following manner: The Web server runs the CGI program on behalf of a user, the CGI program does some processing, creates an output HTML document, and then stores it in an agreed upon filename or memory location. When the CGI program exits, the Web server delivers the document back to the client Web browser. The CGI program may do calculations based on input from the user, or it may access a database and create a document with the requested information. The CGI program can be written in any language supported by the Web server's operating system. For example, the NPS Locator Web server (WebSite 1.1) runs on Windows NT; therefore, any programming language supported by Windows NT can be used for CGI programs. Some common languages may include Visual Basic, C++, Powerbuilder, Java, and Delphi.

One drawback of a traditional CGI executables is their inefficiency when connecting to a database. Each time a user requests a dynamic page of data, the Web server creates a new CGI process, connects to the data source, receives the data, closes the connection and shuts down the CGI process. The larger the CGI program, the more overhead is created on the Web server, resulting in slower performance. In addition, much overhead is incurred from opening and closing a database connection.

To address some of the inefficiencies of CGI, most Web server vendors provide an application programming interface (API), which allows direct access to server-side applications that run as part of the Web server. "In most cases this leads to faster and more compact memory-wise equivalents to CGI programs, though no longer portable." (Kohlhepp, 1997) "Functions written to a server's APIs can be multithreaded, therefore, one process can service multiple requests, keeping the memory requirement down." (Kohlhepp, 1997) CGI, on the other hand, must start a new process for each request.

Though APIs can have better performance than CGI, APIs can crash a web server because of the shared web server memory space. CGI extensions are separate programs that cannot affect the operation of the web server itself. Web applications that use the API extensions approach are much more complex and demand advanced programming skills such as "multithreading and concurrency synchronization, network protocol programming and structured exception handling." (Denny, 1996)

It is important to note that both approaches provide the same functionality within the application architecture, the only difference being the issue of performance. Both approaches still require opening and closing of data connections.

Furthermore, both methods fail in the area of state management. HTTP is a *stateless* protocol. Each browser request is distinct, and in its raw form, completely disassociated from any prior request. This works well for static HTML pages, but does not work in a database application for transaction management or navigation. The server needs to know who performed what operation in order to commit a transaction, or needs to know what record the user is on to know which record is the next one. By the nature of the Web, there is never a constant connection to the Web server, so this information is not retained. (Krull, 1997)

Many vendors have developed an architecture to solve the issues of CGI performance, persistent data connections and state management. This architecture, as seen in Figure 1, has four main components: a courier module, a broker, an application agent, and a database. This architecture is based on the architectures from Borland, NetDynamics, and HREF Systems. The architecture has been generalized and some of the component names changed to simplify the discussion.

Application requests flow through the architecture in the following manner:

1. The Web server passes a browser's HTTP request to the broker module

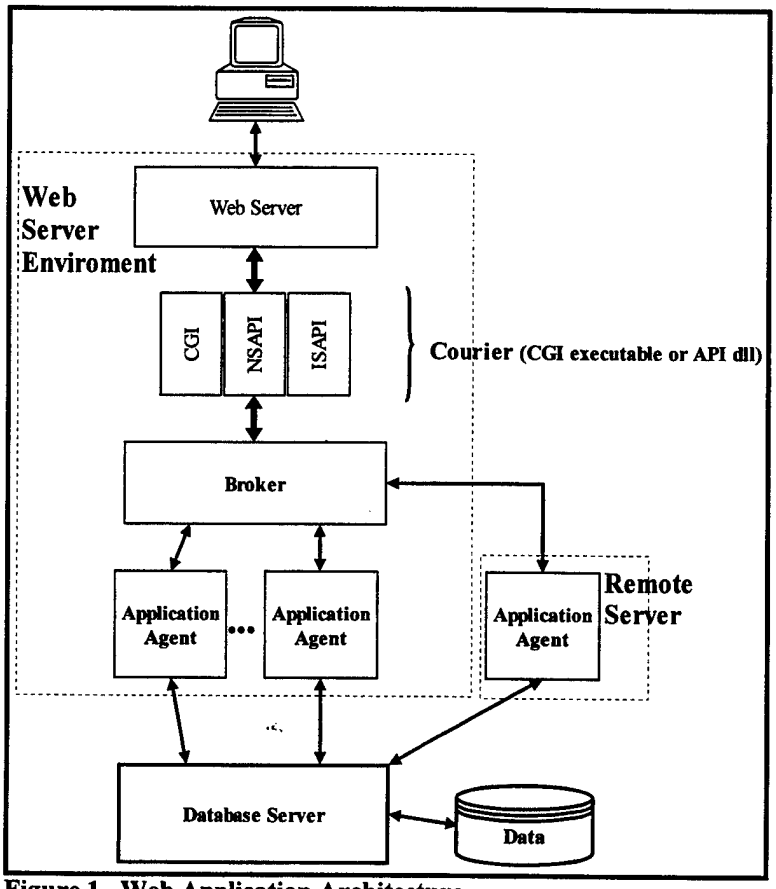


Figure 1. Web Application Architecture

through a courier, which is either a lightweight CGI "stub" (15-50kb for most implementations), or a Web Server API-based (NSAPI or ISAPI) dynamic link library (DLL).

2. The broker receives the request from the courier and then passes it to the appropriate application agent. The broker is responsible for assigning a unique session ID to new clients so that user "state" can be maintained across multiple requests from the same user. This ID is included in every HTML page generated, so that when the user makes a subsequent request, they will

pass back the session ID with their request, allowing the Application Agent to track the user.

3. The Application Agent then processes the request by accessing a database and applying any necessary business logic. The Application Agent then creates the output HTML document and passes it back to the Web server and the Courier module shuts down, signaling the Web server to deliver the document.

In this architecture, the Application Agents stay loaded and can maintain data connections, eliminating the performance problem with traditional CGI architecture. In addition, the reduced size of the CGI program also saves in resources and performance. In this architecture, the CGI is used merely to pass information to the backend system and to signal when the HTML document is ready for transmission back to the user.

Most vendors allow there to be more than one instance of an Application Agent to be running at a time. The Broker is responsible for load-balancing requests across the Agents. In addition, most systems allow the Agents to be located on different workstations, therefore allowing for scalability of the system by adding additional remote servers as seen in the architecture diagram.

III. NPS LOCATOR DEVELOPMENT

A. REQUIREMENTS

In developing the NPS Locator Prototype, one guiding principle existed: The prototype should require minimal changes to the NPS computing environment. The computing environment refers to the data sources, network infrastructure, and client machines. The reason for this requirement was to eliminate any barriers, political or technical, to implementing the system. Then, after the prototype has been implemented and has proved its value, requests for changes to the environment may be made.

The remaining portion of this section presents the requirements defined for the design and functionality of the NPS Locator prototype. These requirements are limited in scope because of the prototype nature of the Locator and do not represent the "ideal" directory system.

1. Functional Requirements

- *The system must provide a means for NPS personnel to obtain communication and location information on any other person at NPS.* This information includes email address, phone and fax numbers, building and office locations, homepage URLs, mail codes, position title, and a picture. Any information that aids, either directly or indirectly, the location of or eventual communication with a particular individual should be included.

- *The system must provide search capabilities to locate personnel by last name, department, and status (i.e., student, staff, or faculty).* The last name search should provide partial name searches in which the user is only required to input the first few letters of the last name.

- *The system should allow users to augment their locator listing with information they provide through the system.* Users should be provided with the option of providing other information that enables communication with that user. For example, users may want to input their home phone and mailing address, or they may want to provide a time schedule of when they are available for phone calls and appointments.

- *The system should maximize the use of built-in communication facilities.* For example, if using web browser technology, email addresses should be presented as a hypertext link that when selected will call up the browser's

built-in mail tool. Also, hyperlinks should be used when possible to link to additional information or resources.

- *The locator should maximize automatic methods of updating the locator data stores, and minimize human intervention.* Manual updates often do not get done on a regular basis because they can be very tedious. Automatic updates are easily run on a much more frequent basis, which results in more current data.

2. System Requirements

- *The NPS Locator should be accessible from multiple platforms to include UNIX workstations, PCs, and Macintosh computers.*
- *The system should require little or no additional software on the client machines except for a web browser.*
- *The system should be accessible from any network that supports TCP/IP and HTTP.* NPS currently has four major network operating systems (NOS) installed: UNIX TCP/IP, Banyan Vines, Novell NetWare, and AppleTalk. Currently, all these NOS support TCP/IP.
- *The NPS Locator should be developed with standard tools that are familiar to the NPS computing support staff.* These tools include primarily Borland Delphi and Oracle development products.

3. Data Requirements

The data sources for the NPS Locator should be from the data owners to ensure that the most accurate and current information is used. The Locator should either directly access the original data for its queries, or it should build its own data store from the original data. If building its own data store, then the frequency of the updates must be set such that the updates capture most of the changes to the original source.

B. DEVELOPMENT TOOLS

The tools selected for the prototype development were, for the most part, dictated by their familiarity within the NPS computing support staff. The tools, therefore, may not represent the "best" tools for the job, but did allow for a short development time, which is in line with the prototype concept.

1. Programming Tools

The most significant tool to the prototype development was WebHub from HREF Systems. "WebHub is a CGI framework for building dynamic, database-driven web sites with Delphi for deployment on Windows-based web servers." (HREF, 1996) "A Framework is a set of prefabricated software building blocks that programmers can use, extend, or customize for specific computing solutions." (Taligent, 1997) Frameworks prevent a software developer from having to start from scratch each time he writes an application. WebHub's framework includes "over 30 web-specific Delphi components" for building custom made web applications. WebHub uses the Web "broker-agent" architecture discussed above.

Accordingly, Delphi was used in conjunction with WebHub to add the business logic specific to the NPS Locator application. Delphi also was used to build programs that populate the Locator's database. These Delphi programs collect data from various data sources throughout the NPS infrastructure, normalize it, and then store the data in the database.

Perl allowed data aggregation from UNIX platforms. Perl stands for Practical Extraction and Report Language. It is ideal for extracting data from UNIX file systems and then formatting that data into specific output files.

2. Web Server

O'Reilly's WebSite Web Server was chosen to host the NPS Locator prototype because of its wide use at the school. WebSite can run on Intel platforms using either Windows 95 or Windows NT. The WebSite-Windows NT combination served as the NPS Locator Web platform.

3. Database Server

Finally, the NPS Locator database was tested on two platforms. The first was on an Oracle 7.2 database server. The database server platform was a DEC Alpha workstation running an Alpha version of Windows NT. Once again, the selection of this component resulted from its availability and familiarity at NPS. Some expertise existed with the Oracle server because it was used as a test bed for possible migration from the mainframe database to a client-server workstation environment. The second test was run with a local Paradox database resident on the Web server machine. Both test were successful and provided acceptable results.

C. NPS LOCATOR ARCHITECTURE

The NPS Locator has two main components: the NPS Locator Web Application and the Locator Update component. The Web Application provides the user interface, business logic, and data representation. The Update component gathers data from a variety of data sources, parses that data, normalizes it, and then stores it in the Locator database. This section will first discuss the Update component and then follow with an explanation of the Web Application.

1. Update Component

The Update component is a collection of data sources, automated data-extraction scripts, and file-transfer routines. These various components can be seen in Figure 2 below.

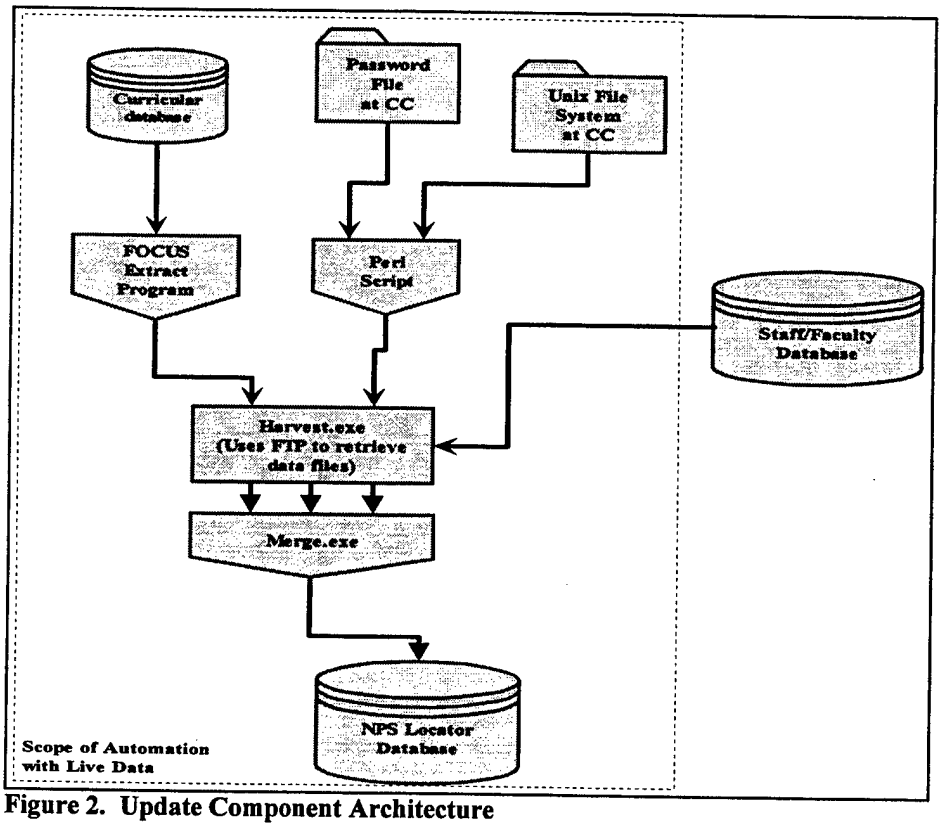


Figure 2. Update Component Architecture

The Update component can be broken down into three phases. Phase I consists of the extraction of student data from the Curricular Database, extraction of email and homepage URL information from the Computer Center (CC) UNIX accounts, and extraction of staff and faculty information from the Staff/Faculty Database. Phase II

consists of "harvesting" the extracts created in Phase I, and storing them on the Web Server. Phase III merges the extracts and then stores them in the Locator's Oracle Database Server. Before discussing the three phases, an explanation is provided of how the data sources were selected for the prototype and what information they provide.

a. Data Sources

One of the goals of the prototype is to eliminate, or at least minimize, the duplication of effort in creating and maintaining data sources. This is accomplished by using the data sources of the data owners. A "data owner" is, for the purposes of this thesis, a person or organizational body that is responsible for the creation and maintenance of particular data, usually relevant to their organizational function. For example, the Registrar is the "data owner" of all information pertaining to grades and class schedules. The Registrar creates and maintains a "Registrar Database" that contains data that pertains to grades and class schedules. Therefore, one would expect that the best source (i.e., the most accurate and up-to-date) for such information would be the Registrar Database.

Using data ownership as the primary criterion in selecting data sources, the Curricular Officer's Database was selected as the data source for students' personal information and the CC UNIX user accounts were selected as the data source for email and homepage URL information. The information extracted from the Curricular Database is as follows: SSN, name, grade, service, student mailbox number, department, curriculum, section, entry date, and graduation date.

The CC UNIX password file and file system serve as the second data source used by the Update component. A sample line entry in a UNIX password file is shown below:

```
jeforte:67kBjs/mctYuc:24865:20:Jeffrey E. Forte, 2513:/h/joshua_u2/jeforte:/bin/csh
```

Each field in the password file is separated by a colon (":"). The fields are:

- *username*, which also serves as the email address ("jeforte")
- *password*, which is the user's encrypted password
- *userid*, which is the unique (to the UNIX system) numerical ID that represents this user
- *groupid*, which is the ID of the group to which this user belongs

- *quota* and *comment*, which represent different things on different machines
- *infofield*, which is a character string containing personal information about the user. In the case of CC accounts, it contains the user's full name, followed by a comma, and a unique four digit number used by the administrator to track the user.
- *homedir*, which is the user's home directory
- *shell*, which is the command shell that is started when the user logs in

The information extracted from the password file includes last name, first name, middle initial, email address, and home directory. The home directory is extracted in order to determine if a user has a homepage. If a user of the CC UNIX system wants to have a homepage, the user is required to create a directory called "public_html" under his or her home directory, and place the page in this directory. The file must be named "Welcome.html" or "Index.html." By searching the home directory, it can be determined if the user has a homepage by searching the user's home directory.

The final data source used is a staff and faculty database that was created for use by the prototype. It was difficult to find good data sources for staff and faculty information, and the ones that did exist, were not readily available because of bureaucratic reasons. Again, since the staff and faculty data is relatively static compared to the student data, for the purpose of the prototype this database was sufficient. The proof in concept can be demonstrated on the more dynamic student data.

b. Phase I

In Phase I, automated scripts extract information from the data sources discussed above, and then store the data as ASCII flat-files on the respective systems in accounts set aside for the Locator system. Refer to Figure 2. Starting from the left, the first data extraction routine encountered is the "FOCUS Extract" program. This program was written for the prototype by Judy Harr, a FOCUS database administrator. It extracts student information from the Curricular Officer's Database, formats it into an ASCII flat-file, and stores it in a special mainframe account set aside for the Locator prototype. The frequency of this process is adjustable, and was set to process once a day, early in the morning when processing levels are low.

The second data extraction routine consists of a Perl program residing on CC UNIX system. This program parses the password file, and pulls out each username (email address), the user's full name, and his or her home directory. When it gets the

home directory, it checks to see if a "public_html" directory exists. If so, it then looks for a filename of either "Welcome.html" or "Index.html." At this point it writes out to an ASCII flat-file with the following format: last name, first name, middle initial, last name with initials appended, homepage URL (if one exists), and username. The text below shows two entries in the resulting file:

```
FORTE   JEFFREY E. FORTEJE  http://vislab-www.nps.navy.mil/~jeforte  jeforte
EMERY   JAMES   C. EMERYJC                                           jcemery
```

The Perl program is executed once a day early in the morning. This is accomplished with the UNIX "cron" command, which executes commands at specified dates and times.

The third routine would be the extraction of staff and faculty information from the staff and faculty database. Since this database was created for the prototype specifically, it is conveniently located on the Web server ready for the Phase III process, and therefore no real extraction is taking place. If this database did not exist, the third data extraction routine would probably pull information from the Human Resources database as well as other pertinent sources of staff and faculty information. This simulated third routine concludes Phase I.

c. Phase II

Phase II can be considered the "harvesting" phase. In this phase, all of the extraction files are retrieved from their originating locations and stored on the Locator Platform (Windows NT server) in preparation for Phase III. Appropriately, Phase II is accomplished by a program written in Delphi named "Harvest.exe." This program uses an FTP component from Netmanage Corporation. FTP is a file transfer protocol that works on top of TCP/IP. It is useful in transferring files across a TCP/IP network, such as the NPS Intranet. The Harvest program, which is resident on the Locator Windows NT Server, initiates file transfers with the Locator account on the CC UNIX system to retrieve the email/homepage ASCII flat-file, and with the Locator account on the mainframe to retrieve the Curricular Database extract. The Harvest program is setup as a Windows NT service, which allows it to be scheduled to execute at predetermined times. In this case, it is scheduled to run everyday, approximately two hours after the extraction scripts execute. Once the files are residing on the Locator platform, Phase II has successfully completed. At this point, the completion of the Harvest program triggers Phase III.

d. Phase III

Phase III of the Update component takes the ASCII flat-files and the Staff/Faculty database and merges them together, and then stores the resulting data in the Locator Oracle database. The "Merge.exe" program, which is triggered by the completion of the Harvest program, is the workhorse of the third phase. The Merge program also was created using Delphi. The Delphi TBatchMove component is used to convert the ASCII files into the standard DBF format that Delphi supports. This component also ensures no duplicate index keys get added to the resulting database. The Merge program uses this function with the last-name, first-name, middle-initial combination to eliminate duplicate records for all personnel. Then each of the converted databases is cleaned and standardized. For example, the Curricular database file was inconsistent on how it handled two-part last names such as Van Horn and Van Steenberger. In one case, the two parts are appended, as in Vansteenberger. In the other case, the two are separated by a space, as in Van Horn. This inconsistency makes it difficult to merge the UNIX extract file with the Curricular extract file because in the UNIX file Vansteenberger is Van Steenberger. It is unfortunate that names had to be used as the index key for the merge, but the UNIX accounts do not record any part of the SSN or other usable unique identifier. Moreover, the Curricular database does not store separately the last name, first name, and middle initial of a student. They all are combined in a single field labeled "name." As a result, the Merge program does a "best effort" in correlating data from the various data sources using the components of a person's name. For the most part, it is successful.

After cleaning the data, the Merge program uses an SQL "Insert" query to update the Locator's Oracle database resident on a DEC Alpha. In essence, the Locator's main table is recreated every morning. Following the rebuild of the main table, another query is executed that links records in the main table with the "register" table which stores information on users who have registered with the Locator Application. In short, the register table stores the login, password, and other information with which the user chose to augment his listing. At this point, the Merge program exits, and the Update component has completed its function.

2. NPS Locator Application

The Locator Web application architecture can be seen in Figure 3. The application, as discussed earlier, uses WebHub, and therefore utilizes the broker/agent architecture. The application resides on a Windows NT Server. The main components on the NT Server include WebSite Web Server, WebHub, and NPSD.exe (Customized WebHub WebApp). The business logic and Locator functionality are coded into the NPSD.exe Delphi program. It was developed using the WebHub Web Application framework.

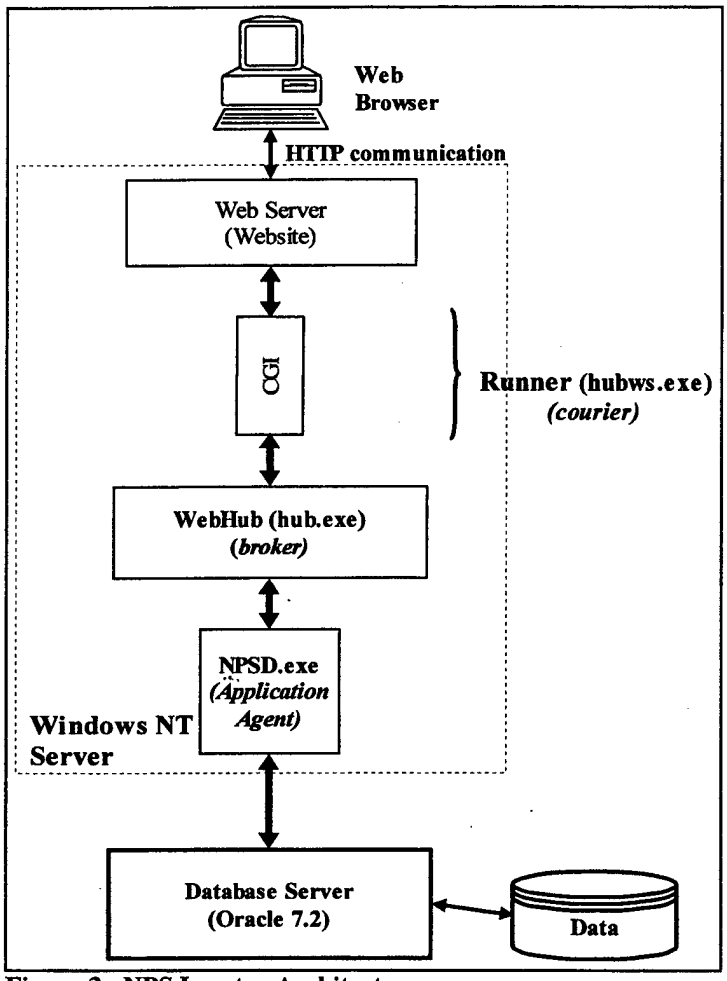


Figure 3. NPS Locator Architecture

The other major component is the data store for the Locator data. The data resides on a DEC Alpha workstation running Oracle 7.2 Database Server, with two main tables, one labeled "main" and the other labeled "register." The structure of these tables can be viewed in Appendix A. The Locator NT Server communicates with the Oracle Server via

the TCP/IP NPS Intranet. More specifically, the NPSD program uses Borland's SQL Links to communicate with the Oracle database. SQL Links in conjunction with Oracle drivers allows full connectivity to the Oracle Database Server from any Delphi program utilizing Borland's Database Engine (BDE).

D. FUNCTIONAL DESCRIPTION

This section describes the functional operation and capabilities of the NPS Locator prototype. When possible, various screen-shots of the prototype will be presented to aid in the description of whatever aspect is being discussed at the time. The first screen-shot is shown below in Figure 4. It is the main page of the Locator. The options available to the user at this point are "search," "login," and "register." Each will be discussed below.

1. Search

The search feature represents the real purpose of the directory system. The Locator provides a basic search capability. The user can search for a person in a few different ways, usually depending on how much information the user already has about the person he is looking for.

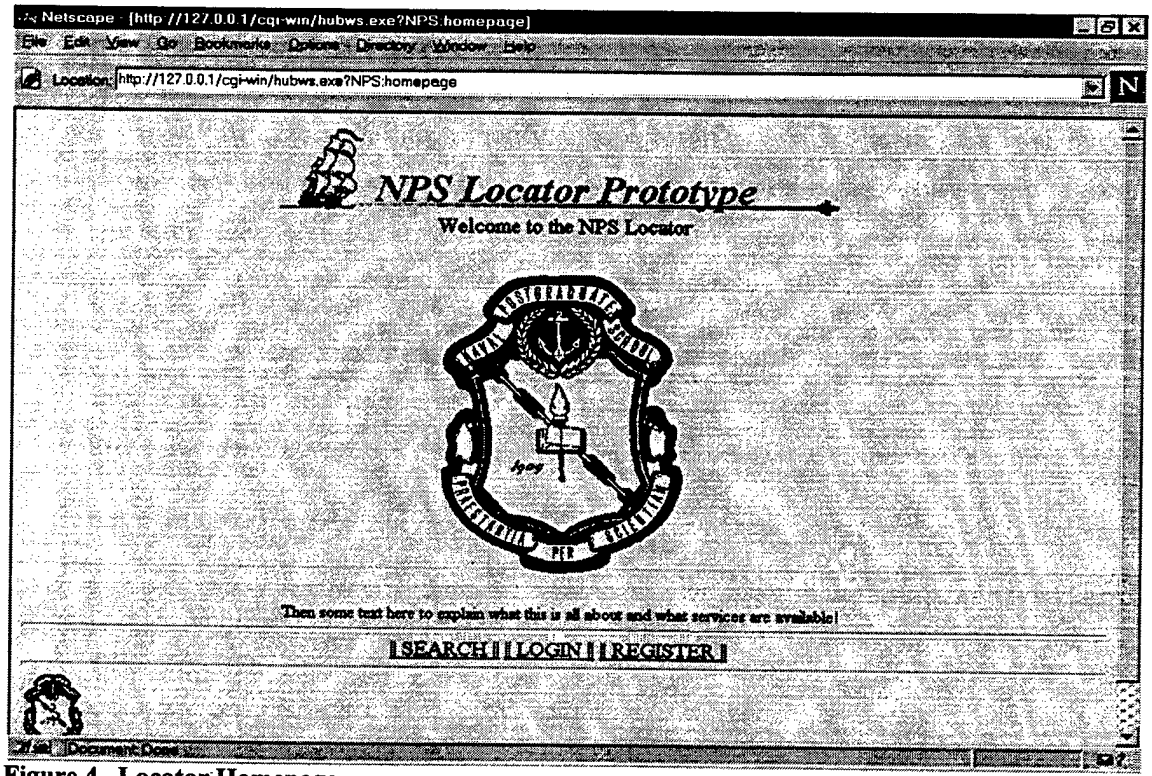


Figure 4. Locator Homepage

Figure 5 shows the search page for the Locator. The user has the option to use a "first letter" search by selecting the appropriate hyperlinked letter in the displayed alphabet. This will generate a query to the Locator database, which will return all personnel with a last name starting with the selected letter. The user then can scroll through the records to find the name he was looking for.

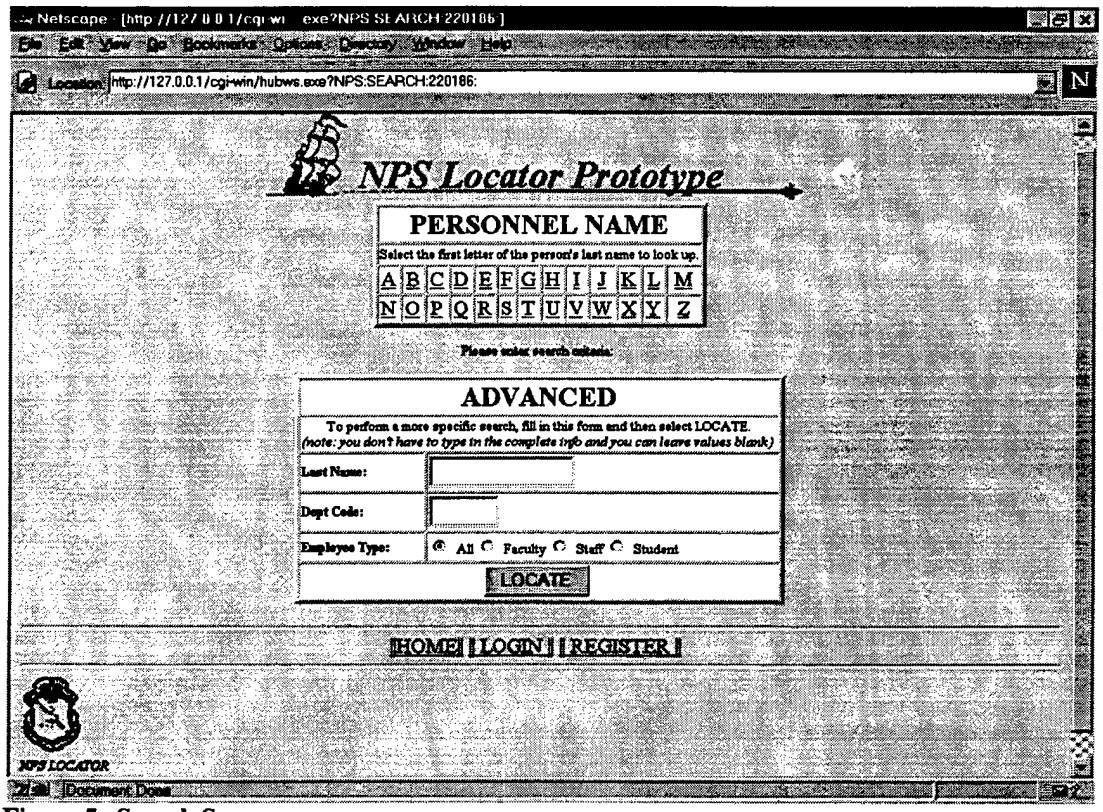


Figure 5. Search Screen

For a more narrow search, the user can type in all or part of the last name in the appropriate block in the Advanced search dialogue. The search is not case-sensitive. The user can further narrow the search by entering in a department code and/or the employee type. The available employee types are faculty, staff, and student. The department code should later be implemented as a selection box, so that users can select the department code without confusion of what should be entered, and to prevent spelling errors.

Once a user a has initiated a search, the Locator will respond with the search results in a table format as shown in Figure 6. The information includes last name, first name, middle initial, title, department, mail code, phone and fax number, email, and a homepage link. If the person has an email account at the computer center, then his or her email will show up in the column, and it will be hyperlinked. The hyperlinked email will, when selected, bring up the user's mail program with the "to:" block filled in with the email address selected from the Locator. If a person has a homepage at the computer

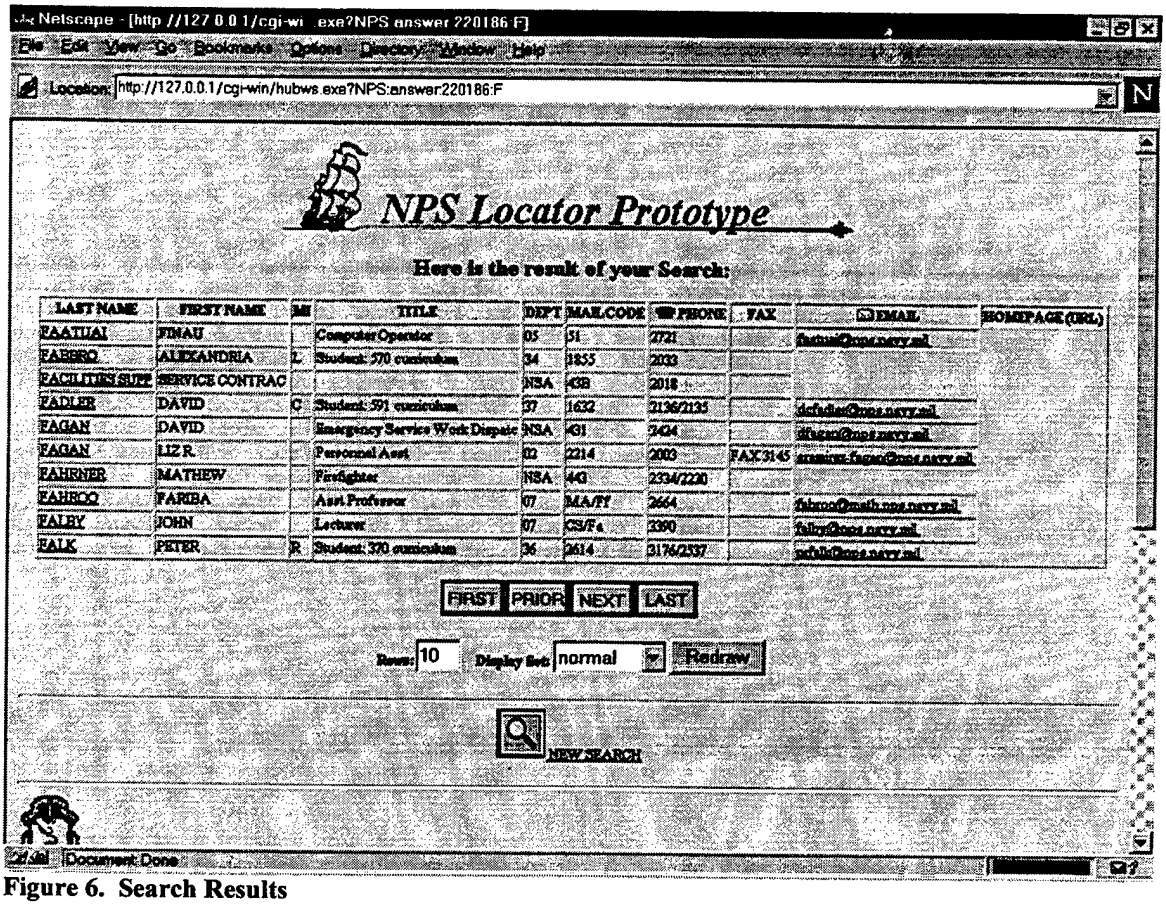


Figure 6. Search Results

center, then his or her homepage URL will be listed under the "Homepage" column and will be hyperlinked.

The last name column is hyperlinked also. When it is selected, it displays a page that provides more detailed information on the particular person. If the person has registered with the Locator, then there may be additional information including a picture, if the person chose to provide it. Figure 7 shows what the detail page looks like.

2. Register

NPS personnel can "register" with the Locator by clicking on the "register" link on the home page. This then prompts the user to enter his last name so the Locator can display his record. Once the appropriate record is found, the user selects his record by clicking on the hyperlinked last name. This then prompts the user to enter in an email address, which will serve as their login name. The email address is also used by the Locator to automatically send the user a system-generated password, which, when used in conjunction with the login, allows entry into the Locator.

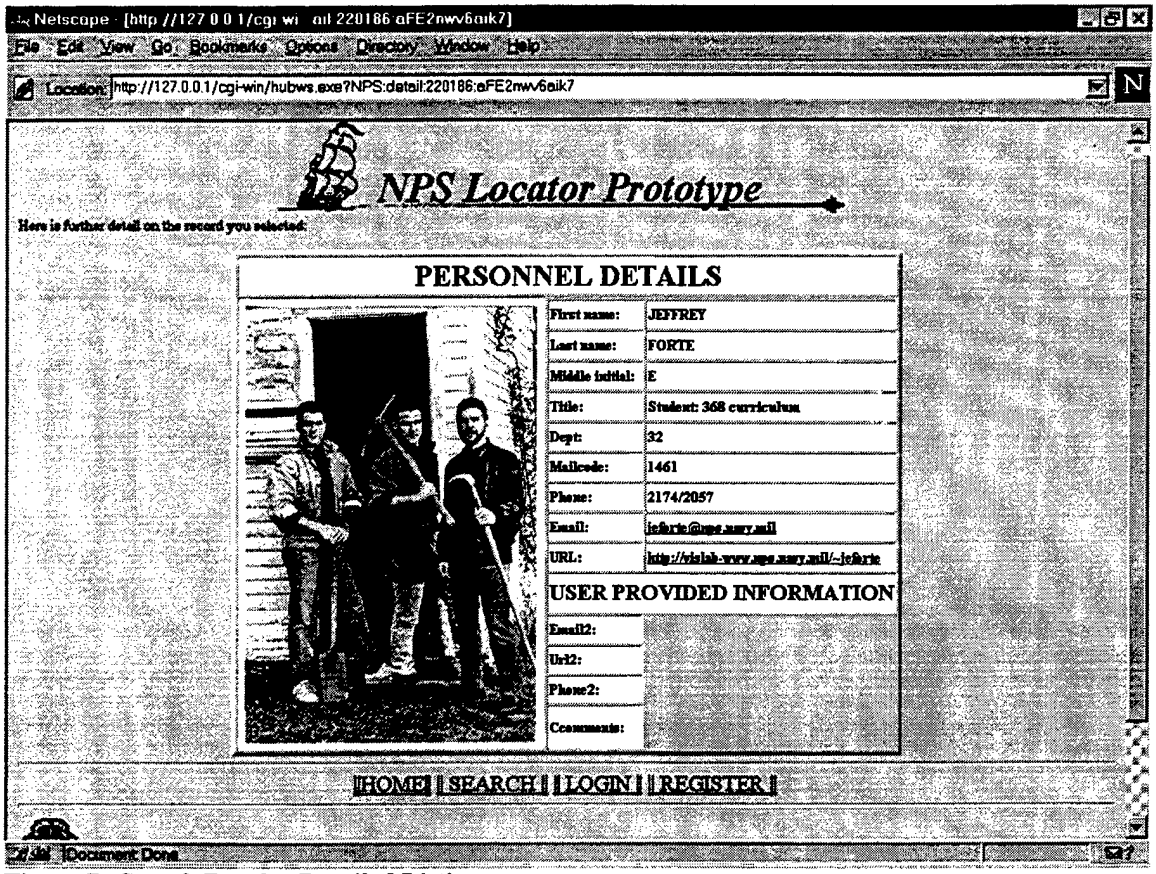


Figure 7. Search Result: Detailed Listing

If a user is not in the system, the user is incorrectly listed in his or her respective data source. For example, a student not listed in the Locator database, in most cases, is not listed in the Curricular Officer's Database. The appropriate action by the user would be to contact his Curricular Office and verify that he is listed correctly in the database.

3. Login

Once a user is registered, he has the ability to augment the system generated information. For the prototype system, a user can upload a picture, and add one additional email address and an additional URL, both of which will be hyperlinked. Finally, the user can add information in a text box, up to 256 characters. This block should be used for information that enables communication with the listed person. For example, a professor may place his class schedule and available officer hours in this block. A staff person may put in the days that he or she plans to take leave. As a final example, a student may put in his thesis study area so that others may locate him.

IV. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The NPS Locator provides a simple, interactive, and easy-to-use facility for users to access useful information about other NPS personnel. The Locator successfully ran for two months during a test in the NPS computing environment. Updates from the Curricular Officers' database and from the CC UNIX accounts were successfully merged and uploaded to the Locator database. This database provided an accurate and up-to-date resource of personnel information that enabled the communication and location of NPS personnel. In addition, the Locator allowed a user to augment his or her system generated information with additional information to further enable communications between individuals.

The implementation of the Locator prototype did not unveil any new revelations for those in charge of computing at NPS, but did re-emphasize some of the current issues NPS is struggling with, as well as reinforce the benefits of Intranet technology. The explosive growth of Web technology and intranet implementations in the Corporate world can serve as a testimonial for the potential of intranets.

The benefits that NPS can obtain from intranet technology are as follows:

- The Web browser is a universal interface that allows interoperability in NPS's multiplatform and multi-NOS environment.
- The thin-client architecture of intranet applications allows NPS to use fewer application development tools and eliminates costly multiplatform deployment of applications.
- Intranets provide a fast and ubiquitous information distribution platform that faculty, staff, and students can use.
- The intranet's ease of updates fosters the publishing of information that is more accurate and up-to-date.
- Intranet technology is relatively cheap which is important in the times of budgetary constraints.

Perhaps the real value of the implementation of the NPS Locator prototype lies in its emphasis of issues plaguing the application of information technology at NPS. For example, an NPS Phone Book application was developed recently, and is appropriately available on the NPS homepage. The directory is complete and is easy to use. Its data

source is a centralized SQL database maintained by code 05. Despite its accuracy and timeliness, many organizational units still have their own Web directory on their organizational homepage. The information in these directories is not maintained in synch with the Code 05 database. This duplication of effort and lack of some centralized management will hinder the success of an effective intranet implementation at NPS.

B. RECOMMENDATIONS

In order to implement any directory system or other intranet application at NPS, many issues must be dealt with. Some of the issues outlined below are indeed addressed in the recently published 1997 draft of the "Strategic Plan for Computing at the Naval Postgraduate School." The issues to be explored are:

- *There is a need for a centralized personnel database that contains all NPS personnel.* Code 05 has developed and maintains a centralized personnel database that currently supports the new accounting system and the NPS Phonebook. The database should further be integrated into other systems as the sole source for personnel information. Policy should be enacted mandating the use of this database as the source for all personnel information and that it be properly maintained as such.

- *All NPS personnel should have an account or alias at the NPS Computer Center.* This provides a guaranteed uniform email address for all personnel at NPS. This coupled with a naming standard will make email addresses derivable just by knowing a person's name. For example, if the standard is a concatenation of the first initial, middle initial, and the first six characters of one's last name, and given that the domain for NPS is *nps.navy.mil* then deriving an individual's full email address is simple. *Jeffrey E. Forte* would be *jeforte@nps.navy.mil.*

- *All organizational entities should have an organizational mailbox.* An organizational mailbox is not associated with any one person within an organization. It allows communication with the organization without having to know someone within that organization, and it serves well as a method for passing official messages. Organizational mailboxes may be best implemented as hotlinks on an organizational chart.

- *Support should be demanded for centralized management of the computing infrastructure that is balanced with distributed control within organizational units.* The Provost of Computing needs to have support from the highest authority at NPS, and a method for enforcing adherence to standards. The right balance between centralized and decentralized control and areas of

responsibilities needs to be explored to maximize the effectiveness of intranet implementation.

- *An NPS data dictionary should be developed for all informational entities and attributes.* Sometimes it is not practical to use one centralized database for all applications because of performance issues, security restrictions, or communication limitations. It would be advantageous to record all information, despite how it is used, in a uniform and predictable manner. For example, many of the databases reviewed as potential databases for the Locator stored the elements of a person's name in very different ways. The Curricular database stored the name as one field, separating the last name and first name by a comma. Other databases had the usual breakout of the name into its three components—first, middle, and last names. The use of a data dictionary allows for any possible future exchanges of information with any information store to be compatible with respect to the format of the individual data elements.

- *Consideration should be given to the creation of a unique identifier for each person at NPS.* This would allow the use of this id as a unique index key to individuals without the possibility of compromising privacy by use of the social security number. Furthermore, any time information is recorded on an individual, password files included, this unique id should be recorded as well.

- *NPS should conduct a survey of commercial directory systems for NPS use.* Many vendors are offering directory systems that allow customization, and most of them provide a single login and password for access to all network resources from anywhere on the network. Banyan's StreetTalk for NT and Novell's NDS for NT are two popular offerings.

Any tool is effective only if used appropriately. Intranets are no exception. Technology is only half of the solution; management is the other half. If a synergy exist between these two elements, an intranet can help harness the potential of an organizations information technology.

LIST OF REFERENCES

Currid, C., "Quit Stalling And Start an Intranet," Windows, April 1997. Available at *http://www.techweb.com/se/linkthru.cgi?WIN19970401S0033*

Denny, R., *SSI, CGI, API, or SSS? Choosing the Right Tool for the Job*, 1996. Available at http://solo.dc3.com

Evans, T., *Building an Intranet*, Sams Publishing, 1996.

HREF Tools Corp., *WebHub Users Manual*, HREF Tools Corp., 1996.

Kohlhepp, R., *Interactive Network Design Manual, Intranets: How To Cut Through The Cob Webs of Internal Information*, 1997. Available at *http://techweb.cmp.com/nc/netdesign/intra5implementing.html*

Krull, K., Intrabuilder Architecture Overview, 1997. Available at *http://www.borland.com/intrabuilder/papers*

Linthicum, D. S, *David Linthicum's Guide to Client/Server and Intranet Development*, Wiley Computer Publishing, 1997.

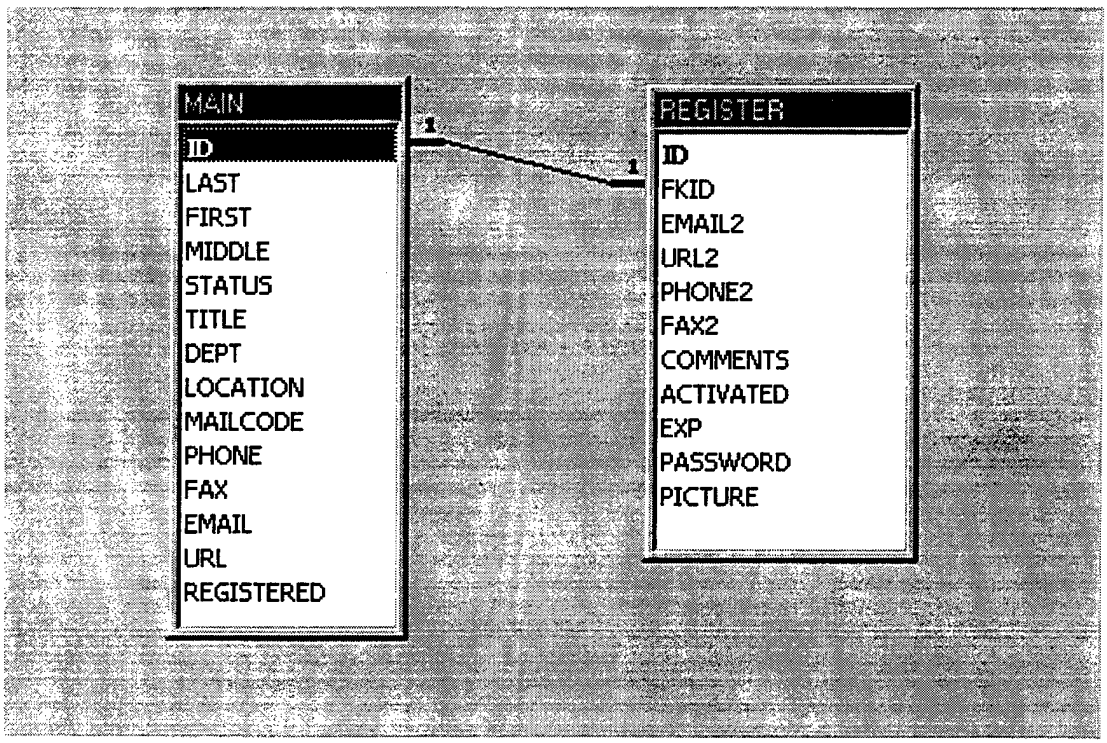
Linthicum, D. S., "Complexity Revisited: Why Web Development is Becoming More Complex than Client/Server," DBMS, April 1997.

Taligent, Inc., Object Resources: Object Glossary, 1997 Available at *http://www.taligent.com*

Telleen Ph.D., S. L., Intranet Methodology: Concepts and Rationale, Amdahl Corp., 1996. Available at *http://orpheus.amdahl.com/doc/products/bsg/intra/concepts1.html*

Till, D., *Teach Yourself Perl 5 in 21 Days, 2nd Edition*, Sams Publishing, Indianapolis, Ind., pp. 553-554, 1996.

APPENDIX A. DATABASE TABLES



(HARVEST Module)

```
unit ftph1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, OleCtrls, NMCT, ExtCtrls, ComCtrls, ISP;

type
  TForm1 = class(TForm)
    FTP: TFTP;
    Timer1: TTimer;
    Status: TStatusBar;
    Label1: TLabel;

    procedure FTPAbort(Sender: TObject);
    procedure FTPExecute(Sender: TObject);
    procedure FTPNoop(Sender: TObject);
    procedure FTPStateChanged(Sender: TObject; State: Smallint);
    procedure Download( const getFile,putFile:string);
    procedure Timer1Timer(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure WaitToAllowProcessing;


  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

var
   LogStr:TStringList;

const
   cRemoteHost    = '131.120.50.206';
   cPort          = '21';
   cUser          = 'locator';
   cPass          = '2nttk4u';
   cFileToGet1    = 'userlist.txt';
   cFileToGet2    = 'currfil1.data';
   cFileToGet3    = 'currfil2.data';
   cDataDir       = 'c:\thesis\npsd\';
   cFileToWrite1  = cDataDir + cFileToGet1;
   cFileToWrite2  = cDataDir + 'curric1.txt';
   cFileToWrite3  = cDataDir + 'curric2.txt';
   cLogDir        = 'c:\temp\';

procedure TForm1.FTPAbort(Sender: TObject);
begin
   LogStr.add('abort event');
   {WANT TO WRITE OUT TO A LOG}
end;

procedure TForm1.FTPExecute(Sender: TObject);
begin
     LogStr.add ( 'execute');
end;

procedure TForm1.FTPNoop(Sender: TObject);
begin
```

```
    LogStr.add('Noop');
end;



procedure TForm1.FTPStateChanged(Sender: TObject; State: Smallint);
begin
  with FTP, LogStr do
    case State of
      prcConnecting   : add('Connecting');
      prcResolvingHost: add('Connecting');
      prcHostResolved : add('Host resolved');
      prcConnected    : add('Connected to: ' + RemoteHost);
      prcDisconnecting: add('Disconnecting');
      prcDisconnected : add('Disconnected');

    end;
end;


procedure TForm1.Download( const getFile,putFile:string);
begin
   Status.simpleText:= 'Downloading ' + getFile + ' ...';
   if (FTP.protocolstate <> 2) then begin
   LogStr.add('Transfer Aborted!');              // do abort recovery...email
                                                 // notification and logging?
   end
   else begin
      WaitToAllowProcessing;
      LogStr.add(inttostr(FTP.protocolstate));
      FTP.GetFile (getFile, putFile);
                        // give it a chance to get info
      WaitToAllowProcessing;

      while (FTP.docoutput.state <> 5) do begin
         application.ProcessMessages;
         if FTP.docoutput.state = 4 then break;
      end;
      LogStr.add(FloatToStr(FTP.docoutput.bytestotal) + ' bytes total transfered to file '
          + putFile);

   end;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
   timer1.interval := 5000;
   timer1.enabled := false;
end;


procedure TForm1.FormActivate(Sender: TObject);
begin
   LogStr := TStringList.create;
   Status.simpleText:= 'Connecting to ' + cRemoteHost + ' ...';
   FTP.connect(cRemoteHost,cPort);
   While (FTP.State = 1) Or
         (FTP.State = 2) Or
         (FTP.State = 3) do Application.ProcessMessages;

   While (FTP.State = 4) And (FTP.ProtocolState = 0) do Application.ProcessMessages;
                                                       // Wait For Connection Response
                                                       // Wait for status change...
                                                       // Check Status Of Control
   Status.simpleText:= 'Authenticating User: ' + cUser + ' ...';
   FTP.authenticate (cUser,cPass);
   //LogStr.add( inttostr(FTP.protocolstate));

   Status.simpleText:= 'Deleting old files...';
   WaitToAllowProcessing;
```

```
   DeleteFile(cFileToWrite1);
   DeleteFile(cFileToWrite2);
   DeleteFile(cFileToWrite3);
   download(cFileToGet1,cFileToWrite1);
   download(cFileToGet2,cFileToWrite2);
   download(cFileToGet3,cFileToWrite3);
   FTP.quit;
   Form1.close;
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
var  F:textFile;
     FileName:string;
begin
    fileName := cLogDir + 'FTP.log';
    AssignFile(F, FileName);
    ReWrite(F);
    Writeln(F, 'FTP Log on ' + dateTimeToStr(now));
    Writeln(F, LogStr.text);
    CloseFile(F);        { Close file, save changes }
end;


procedure TForm1.WaitToAllowProcessing;
begin
   timer1.enabled := true;
   while (timer1.enabled = true) do application.ProcessMessages;
end;
```

(MERGE Module)

```
unit merge1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, DBGrids, StdCtrls, DBTables, DB, libcrypt, ComCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    AsciiTable: TTable;
    BatchMove1: TBatchMove;
    Table2: TTable;
    Table3: TTable;
    Query1: TQuery;
    Query2: TQuery;
    Table4: TTable;
    Database1: TDatabase;
    Main: TTable;
    Query3: TQuery;
    Label1: TLabel;
    Status: TStatusBar;
    Image1: TImage;
    procedure ConvertTxtFilesToDBs;
    procedure CleanCurricData;
    function  RemFirstField(ch:char; var s:string):string;
    procedure MergeCurricWithEmailUrlData;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure OracleBatchUpdate;
    procedure DeleteStudentsFromMain;
    procedure FormActivate(Sender: TObject);



  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

const
   cDataDir = 'C:\Thesis\npsd\';

procedure TForm1.ConvertTxtFilesToDBs;
begin
application.ProcessMessages;
try
  Status.simpleText := 'Converting Curricular Extract File(1)';
  with asciiTable do begin
    close;
    databasename := cDataDir;
    tabletype := ttascii;
    tablename := 'curric2.txt' ;
    active := false;
  end;
  with BatchMove1 do begin
     Source:=asciiTable;
     table2.close;
     table2.databasename := cDataDir;
     table2.tableName:='phase1';
     Destination:=table2;
```

```
      Mode:=batappend;
      AbortOnKeyViol := False;
      KeyViolTableName := cDataDir + 'KEYVIOL1.DB';
      table2.EmptyTable;
      execute;
      table2.close;
      table2.open;
    end;
    Status.simpleText := 'Converting Curricular Extract File(1)';

  //  MessageDlg(IntToStr(batchmove1.MovedCount) + ' records read', mtInformation, [mbOK], 0);
    Status.simpleText := IntToStr(batchmove1.MovedCount) + ' records read! --' +
                         'Now Converting Curricular Extract File(2)';
    application.ProcessMessages;

    with asciiTable do begin
      close;
      databasename := cDataDir;
      tabletype := ttascii;
      tablename := 'curric1.txt';
      active := false;
    end;


    with BatchMove1 do begin
      Source:=asciiTable;
      table3.close;
      table3.databasename := cDataDir;
      table3.tableName:='Phase2';
      Destination:=table3;
      Mode:=batappend;
      AbortOnKeyViol := False;
      KeyViolTableName := cDataDir + 'KEYVIOL2.DB';
      table3.EmptyTable;
      table3.open;
      execute;
      table3.open;
    end;

  //  MessageDlg(IntToStr(batchmove1.MovedCount) + ' records read', mtInformation, [mbOK], 0);
    Status.simpleText := IntToStr(batchmove1.MovedCount) + ' records read! --' +
                         'Now Converting UNIX System Extract File';
    application.ProcessMessages;

    with asciiTable do begin
      close;
      databasename := cDataDir;
      tabletype := ttascii;
      tablename := 'userlist.txt' ;
      active := false;
    end;

    with BatchMove1 do begin
      Source:=asciiTable;
      table2.close;
      table2.tableName:='Phase4a';
      Destination:=table2;
      Mode:=batappend;
      AbortOnKeyViol := False;
      KeyViolTableName := cDataDir + 'KEYVIOL4a.DB';
      table2.EmptyTable;
      //table2.open;
      execute;
      //table2.open;
    end;

  //  MessageDlg(IntToStr(batchmove1.MovedCount) + ' records read', mtInformation, [mbOK], 0);
    Status.simpleText := IntToStr(batchmove1.MovedCount) + ' records read! --' +
                         'Now Eliminating Duplicates in UNIX System Extract';
    application.ProcessMessages;
```

```
  with BatchMove1 do begin
     Source:=Table2;
     table4.close;
     table4.databasename := cDataDir;
     table4.tableName:='Phase4b';
     table4.tabletype := ttparadox;
     Destination:=table4;
     Mode:=batappend;
     AbortOnKeyViol := False;
     KeyViolTableName := cDataDir + 'KEYVIOL4b.DB';
     table4.EmptyTable;
     execute;

  end;
Status.simpleText := IntToStr(batchmove1.MovedCount) + ' records read!';
application.ProcessMessages;
except
  asciiTable.close;
  table2.close;
  table3.close;
  table4.close;
  halt;
end;
end;


procedure TForm1.CleanCurricData;
var
   first,second,third,fourth,temp: String;
   i,size,cnt: integer;
   existcomma: boolean;

begin
   Status.simpleText := 'Cleaning Curricular Extract DATA';
   application.ProcessMessages;
   query1.close;
   query1.open;
   query1.first;

   with Table2 do begin
      close;
      DatabaseName := cDataDir;
      tableName:='phase3';
      EmptyTable;
   end;
   Table2.open;

    while not Query1.Eof do begin
        Table2.append;


        temp := UpperCase(Trim(Query1.fieldbyname('name').asstring));
        size := Length(temp);
        cnt := 0;
        existcomma := false;
        For i := 1 to size do begin
           if temp[i] = ',' then
              existcomma := true;
        end;

        if (existcomma) then begin
           Table2['Last'] := RemFirstField(',',temp);
           temp := trim(temp);
           size := Length(temp);
           For i := 1 to size do begin
              if temp[i] = ' ' then
                 inc(cnt);
           end;


           case cnt of
```

```
        0:  begin
                Table2['First'] := temp;
            end;
        1:  begin
                Table2['First'] := RemFirstField(' ',temp);
                Table2['Middle'] := temp;
            end;

        else  begin
                Table2['First'] := RemFirstField(' ',temp);
                Table2['Middle']:= RemFirstField(' ',temp);
            end;
    end;
end
else begin  //no comma
    For i := 1 to size do begin
       if temp[i] = ' ' then
          Inc(cnt);
    end;

    first  := '';
    second := '';
    third  := '';
    case cnt of
        0:  begin
                Table2['Last'] := temp;
            end;
        1:  begin
                Table2['Last'] := RemFirstField(' ',temp);
                Table2['First'] := temp;
            end;
        2:  begin
                first := RemFirstField(' ',temp);
                second:= RemFirstField(' ',temp);
                third := trim(temp);
                if ((first = 'AL') or (first = 'VAN') or (first = 'VON')) then begin
                   Table2['Last'] := first + second;
                   Table2['First'] := third;
                end
                else begin
                   Table2['Last'] := first;
                   Table2['First']:= second;
                   Table2['Middle'] := third;
                end;

            end;
        else  begin
                first := RemFirstField(' ',temp);
                second:= RemFirstField(' ',temp);
                third := trim(RemFirstField(' ',temp));
                if ((first = 'AL') or (first = 'VAN') or (first = 'VON')) then begin
                   Table2['Last'] := first + second;
                   Table2['First'] := third;
                end
                else begin
                   Table2['Last'] := first;
                   Table2['First']:= second;
                   Table2['Middle'] := third;
                end;
            end;
    end;//case
end; // if-else comma test

Table2['Id'] := copy(crypt(Query1['ssn'], '*$'),3,11)+ copy(Query1['ssn'],9,1);
Table2['Mailcode'] := Query1['mailcode'];
Table2['Dept'] := Query1['Dept'];
Table2['Phone'] := Query1['Phone'];
Table2['Title'] := 'Student:  ' + Query1['curr'] + ' curriculum';
Table2['Status'] := 'STUDENT';
Table2.post;
query1.next;
```

```
      end;
    Table2.close;
    Table2.open;


end;


function TForm1.RemFirstField(ch:char; var s:string):string;
var
   i,size : Integer;
   s2  : String;
begin
   Setlength(s2,255);
   i:= pos( ch,s);
   If i = 0 then begin
      RemFirstField := s;
      Exit;
   end;
   size := (length(s)-1);
   Move(s[i+1], s2[1], size);
   Setlength(s,i-1);
   Setlength(s2,size);
   RemFirstField := s;
   s:= s2;

end;

procedure TForm1.MergeCurricWithEmailUrlData;
var
    I,initcount,fullcount, nomatch,firstcount: integer;
   LookUpResults:variant;
   LuLast,LuFirst,LuMiddle, LuFinit, LuMinit,LuLFM: string;
begin
  // AT THIS POINT WE WANT TO MERGE WITH THE PERL FILE TO GET EMAIL AND URL

    //****************************** Phase 4 merge URL/Email data with Phase3 product
    Status.simpleText := 'Merging Curricular Extract DATA with UNIX Extract DATA';
    application.ProcessMessages;
    Table3.close;
    Table3.databasename := cDataDir;
    Table3.tablename    := 'phase4b.db';
    Table3.open;
    Table3.first;
    Table2.open;
    Table2.first;
    initcount:=0;
    fullcount:=0;
    firstcount:=0;
    nomatch  :=0;
    while not table2.eof do begin
        LuLast  := Trim(Table2.fieldbyname('Last').asstring);
        LuFirst := Trim(Table2.fieldbyname('First').asstring);
        LuMiddle:= Table2.fieldbyname('Middle').asstring;
        LuFinit := copy(LuFirst,1,1);
        LuMinit := copy(LuMiddle,1,1);
        LuLFM   := LuLast + LuFinit + LuMinit;

        if  Table3.Locate('Lname;Fname;MI',
                 VarArrayOf([LuLast,LuFirst,LuMinit]),[loPartialKey]) then begin
           Table2.edit;
           Table2['email'] := Table3['email'] + '@nps.navy.mil';
           Table2['url']:= Table3['url'];
           inc(fullcount);
           table2.post;

        end
        else if Table3.Locate('LastFiMi',LuLFM,[loCaseInsensitive,loPartialKey]) then begin
              Table2.edit;
              Table2['email'] := Table3['email'] + '@nps.navy.mil';
```

```
            Table2['url']:= Table3['url'];
            table2.post;
            inc(initcount);
        end
        else if Table3.Locate('Lname;Fname',
                VarArrayOf([LuLast,LuFirst]),[loPartialKey]) then begin
            Table2.edit;
            Table2['email'] := Table3['email'] + '@nps.navy.mil';
            Table2['url']:= Table3['url'];
            table2.post;
            inc(firstcount);
        end
        else
           inc(nomatch);

        if table2.fieldbyname('middle').asString <> '' then begin
            table2.edit;
            table2['middle'] := copy(table2.fieldbyname('middle').asString,1,1);
            table2.post;
        end;
        table2.next;

    end;

end;


procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  asciiTable.close;
  table2.close;
  table3.close;
  table4.close;
end;



procedure TForm1.OracleBatchUpdate;
var I:char;
begin
   Status.simpleText := 'Updating Oracle Database';
   application.ProcessMessages;
   for I :='A' to 'Z' do begin
      Status.simpleText := 'Updating Oracle Database w/ Last names beginning with:  ' +I;
      QUERY3.CLOSE;
      QUERY3.SQL.CLEAR;
      QUERY3.SQL.ADD('INSERT INTO ":LOC:MAIN" SELECT * ' +
                     'FROM ":NPSD:PHASE3" WHERE LAST LIKE '+ '''' + I + '%''');
      QUERY3.EXECSQL;
      application.ProcessMessages;

   end;
   QUERY2.CLOSE;
   QUERY2.SQL.CLEAR;
   QUERY2.SQL.ADD('UPDATE MAIN M SET REGISTERED=''1'' WHERE M.ID = ' +
       '(SELECT FKID FROM REGISTER R WHERE M.ID=R.FKID )');
   QUERY2.EXECSQL;
   application.ProcessMessages;

end;

procedure TForm1.DeleteStudentsFromMain;
var s:string;
begin
   // The runs a delete query on the Main table...it deletes all student records.
   // The query.sql.text = ' DELETE FROM "MAIN"  WHERE "Status" = 'STUDENT' '
   Status.simpleText := 'Deleting Old Student Data';
   application.ProcessMessages;
   QUERY2.CLOSE;
   QUERY2.SQL.CLEAR;
   Query2.sql.add('DELETE FROM MAIN WHERE STATUS=''STUDENT''');
```

```
   Query2.execsql;
   query2.close;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
   ConvertTxtFilesToDBs;
   CleanCurricData;
   MergeCurricWithEmailUrlData;
   DeleteStudentsFromMain;
   OracleBatchUpdate;
   Application.Terminate;
end;


end.
end.
```

APPENDIX C. NPS LOCATOR MAIN SOURCE CODE MODULE

This Appendix contains the main module of the NPS Locator Prototype. This module provides all the business logic and Web application functionality.

```
unit NPSf;

interface

uses
  SysUtils, Messages, Classes, Graphics, Controls, Forms, Dialogs
, Buttons, StdCtrls, Menus, ExtCtrls
{$IFDEF WIN32}
, Windows
{$ELSE}
, WinProcs, WinTypes
{$ENDIF}
, Toolbar
, TpMenu, TpMemo, WebMemo, Combobar
, WebTypes, WebVars, WebBase, WebCore, WebApp, WebCall
, Webmenu, HtmlBase, HtmlCore, HtmlSend, WebInfo, WebBrows,
  UpdateOk, WebIniFL, WebHttp, WebServ, IniLink, Webconst
  ,WebForm, Restorer, TpApplic, TpLabel, WebLink, WebPage, WebPHub, DB,
  DBTables, WdbLink, WdbScan, WdbGrid, WdbSorce, DBCtrls, Grids, DBGrids,
  TblRetry, Webtbl, WebMail, WebSock, ucstring;

type
  TWebAppNPS = class(TWebAppForm)
    StatusBar1: TtpStatusBar;
    StatusPanel1: TtpStatusPanel;
    WebAppOutput: TWebAppOutput;
    WebInfo: TWebInfo;
    WebIniFileLink: TWebIniFileLink;
    WebServer: TWebServer;
    WebBrowser: TWebBrowser;
    WebCommandLine1: TWebCommandLine;
    WebAppNPS: TWebApp;
    WebSession0: TWebSession;
    tpComponentPanel1: TtpComponentPanel;
    tpAppRole1: TtpAppRole;
    FormRestorer1: TFormRestorer;
    DoMenuBarFile: TtpDfmMenuItem;
    DoExit: TtpDfmMenuItem;
    DoMenuBarHelp: TtpDfmMenuItem;
    DoContents: TtpDfmMenuItem;
    DoTopicSearch: TtpDfmMenuItem;
    DoHowtouseHelp: TtpDfmMenuItem;
    DoTtpDfmMenuItem: TtpDfmMenuItem;
    DoAbout: TtpDfmMenuItem;
    DoActionComponents: TtpMenuItem;
    DoWebPages: TtpMenuItem;
    WebMenu1: TWebMenu;
    tpComboBar1: TtpComboBar;
    ToolBar2: TtpToolBar;
    DoMenuItemView: TtpMenuItem;
    DoMenuItemTool: TtpMenuItem;
    DoMenuItemVerb: TtpMenuItem;
    DoMenuItemIdle: TtpMenuItem;
    tpToolButtonBrowser: TtpToolButton;
    tpToolButtonMemo: TtpToolButton;
    tpToolButtonActivated: TtpToolButton;
    tbBlankPage: TtpToolButton;
    tbUserPage: TtpToolButton;
    tbMemoPage: TtpToolButton;
    Notebook: TNotebook;
```

```
WebHtmlMemo1: TWebHtmlMemo;
tpLabel1: TtpLabel;
tpLabel2: TtpLabel;
SurferTbl: TTable;
DataSource1: TDataSource;
WebDataSource1: TWebDataSource;
Query1: TQuery;
BatchMove1: TBatchMove;
registertbl: TTable;
NPS_ANSWER: TWebPage;
NPS_DETAIL: TWebPage;
DBGrid1: TDBGrid;
DBNavigator1: TDBNavigator;
wgrid1: TWebDataGrid;
NPS_ADMIN: TWebPage;
Edit1: TEdit;
Memo1: TMemo;
wgrid2: TWebDataGrid;
NPS_ANSWERREG: TWebPage;
NPS_DETAILREG: TWebPage;
WebSock1: TWebSock;
WebMail1: TWebMail;
NPS_FBREGISTER: TWebPage;
DataSource2: TDataSource;
NPS_DETAILLOG: TWebPage;
main: TTable;
NPS_FBUPDATE: TWebPage;
NPS_HOMEPAGE: TWebPage;
NPS_LOGINPG: TWebPage;
NPS_FBUPLOADPIC: TWebPage;
NPS_UPLOADPIC: TWebPage;
NPS_LOGUPDATE: TWebPage;
NPS_CHGPASS: TWebPage;
NPS_FBCHGPASS: TWebPage;
npsd: TDatabase;
procedure SelectPage(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure Query1AfterOpen(DataSet: TDataSet);
procedure Grid1HotField(Sender: TWebDataScan; aField: TField;
  var s: string);
procedure Grid2HotField(Sender: TWebDataScan; aField: TField;
  var s: string);
procedure Query1BeforeOpen(DataSet: TDataSet);
procedure NPS_ANSWERSection(Sender: TObject; Section: Integer;
  var Chunk, Options: string);
procedure WebAppNPSEventMacro(Sender: TWebOutputApp; const aMacro,
  aParams, aID: string);
procedure NPS_DETAILSection(Sender: TObject; Section: Integer;
  var Chunk, Options: string);
procedure NPS_ANSWERREGSection(Sender: TObject; Section: Integer;
  var Chunk, Options: string);
procedure NPS_DETAILREGSection(Sender: TObject; Section: Integer;
  var Chunk, Options: string);
procedure NPS_FBREGISTERExecute(Sender: TObject;
  var Continue: Boolean);
procedure NPS_DETAILLOGSection(Sender: TObject; Section: Integer; var Chunk,
  Options: string);
procedure NPS_DETAILLOGExecute(Sender: TObject; var Continue: Boolean);

procedure NPS_FBUPDATEExecute(Sender: TObject; var Continue: Boolean);
procedure NPS_HOMEPAGEExecute(Sender: TObject; var Continue: Boolean);
procedure NPS_LOGINPGExecDone(Sender: TObject);
procedure NPS_HOMEPAGESection(Sender: TObject; Section: Integer;
  var Chunk, Options: string);
procedure NPS_FBUPLOADPICExecute(Sender: TObject;
  var Continue: Boolean);
procedure NPS_UPLOADPICExecute(Sender: TObject; var Continue: Boolean);
procedure NPS_LOGUPDATESection(Sender: TObject; Section: Integer;
  var Chunk, Options: string);
procedure NPS_LOGUPDATEExecute(Sender: TObject; var Continue: Boolean);
procedure NPS_FBUPLOADPICSection(Sender: TObject; Section: Integer;
```

```
      var Chunk, Options: string);
    procedure NPS_CHGPASSExecute(Sender: TObject; var Continue: Boolean);
    procedure NPS_FBCHGPASSExecute(Sender: TObject; var Continue: Boolean);

  private
    { Private declarations }
    function getNextDBFilename:String;
    procedure SetSurferDb(Tbl:TTable);
    procedure SetDBEnv(Tbl:TTable);
    function GeneratePassword :string;
    function GetLiteral(LiteralName :string):string;
    function GetFormLiteral(LiteralName :string):string;
    Procedure PutLiteral(LiteralName,LiteralValue:string);
    procedure BounceTo(PageID:string);
    {upload utilities}
    procedure UpLoadUserFiles;
    procedure SaveUploadFile(const UploadInfo:string;
      var NewFileName,ClientFileName:String);
    procedure ForceFileRename(const oldName,newName:string);
  public
    { Public declarations }
  end;

var
  WebAppNPS: TWebAppNPS;

implementation

{$R *.DFM}

{-------------------------------------------------------------}
var
  nextTempNum:integer;      { used to cycle thru temp table names }
  initFlag: Boolean;        { flag for use with temporary workaround }

const
  maxSimulQueries=25;
  cPageIdFrontDoor = 'homepage';
  cWebSvrDir = 'C:\Website\htdocs\';
  cImageDir =  cWebSvrDir + 'npsd\img\';
  cDataDir  = 'C:\Thesis\npsd\';

{-------------------------------------------------------------}

procedure TWebAppNPS.FormCreate(Sender: TObject);
begin
  SelectPage(tbBlankPage);

end;

{-------------------------------------------------------------}

procedure TWebAppNPS.SelectPage(Sender: TObject);
begin
   with TtpToolButton(Sender) do begin
      NoteBook.PageIndex:=Tag;
      Down:=True;
   end;
end;

function TWebAppNPS.getNextDBFilename:String;
begin
   inc(NextTempNum);
   if NextTempNum>maxSimulQueries then NextTempNum:=1;
   result:='qry'+intToStr(NextTempNum)+'.db';
end;

procedure TWebAppNPS.Query1AfterOpen(DataSet: TDataSet);

begin
   with BatchMove1 do begin
```

```
      Source:=Query1;
      SurferTbl.close;
      SurferTbl.tableName:=getNextDBFilename;
      Destination:=SurferTbl;
      Mode:=batCopy;
      execute;
      query1.close;
      SurferTbl.open;
   end;
   WebAppNPS.Literal['DBName'] := SurferTbl.tablename;
   SetDbEnv(SurferTbl);
end;

procedure TWebAppNPS.Grid1HotField(Sender: TWebDataScan;
  aField: TField; var s: string);
var
   desc : string;
   g:TWebDataGrid;
   tc:TDataSet;  { tc stands for Table Cursor, a bit of a trad from ObjectPal }
   id:String;
begin
  desc := aField.asString;
  g:=TWebDataGrid(Sender); { so we can reuse this proc for multiple grids }
  tc:=aField.DataSet;
  id:=tc.fieldByName( 'Id' ).asString;
  {JUMP is a built-in WebHub macro. Syntax is: JUMP|pageId,command|visiblePhrase }
  if compareText(aField.fieldName, 'Email')=0  then
     s:='<TD><A HREF=mailto:' + afield.asstring + '>' + afield.asstring
        + ' </A>/TD>'
  else if compareText(aField.fieldName, 'Last')=0  then
          s:='<TD>' + '%=JUMP|detail,'+ id + '|' + desc +'=%'
  else
          s:= '<TD><A HREF = ' + afield.asString + '>' + afield.asString + '</A>/TD>';

end;


procedure TWebAppNPS.Grid2HotField(Sender: TWebDataScan;
  aField: TField; var s: string);
var
   desc : string;
   g:TWebDataGrid;
   tc:TDataSet;  { tc stands for Table Cursor, a bit of a trad from ObjectPal }
   id:String;
begin
  desc := aField.asString;
  g:=TWebDataGrid(Sender); { so we can reuse this proc for multiple grids }
  tc:=aField.DataSet;
  id:=tc.fieldByName( 'Id' ).asString;
  {JUMP is a built-in WebHub macro. Syntax is: JUMP|pageId,command|visiblePhrase }
  if compareText(aField.fieldName, 'Last')=0  then  begin
     if  not(tc.fieldbyname('Registered').value = '1') then
        s:='<TD>' + '%=JUMP|detailreg,'+ id + '|' + desc + '=%'
     else
        s:='<TD>*' + desc;
  end
  else
      s:= '<TD><A HREF = ' + afield.asString + '>' + afield.asString + '</A>/TD>';
end;


procedure TWebAppNPS.Query1BeforeOpen(DataSet: TDataSet);
var  srchkeys:integer;
     tok:string;
begin
      tok := '%';
      srchkeys := 0;
      with TQuery( DataSet ) do begin
           sql.clear;
           sql.add( 'SELECT * FROM MAIN' );
```

```
          if WebAppNPS.WebServer.FormLiterals.Values['Last'] <> '' then begin
             sql.add( 'WHERE Last LIKE ');
             sql.add('''' +  uppercase(WebAppNPS.WebServer.FormLiterals.Values['Last'])+ tok  +
'''');
             inc(srchkeys);
          end;
          if WebAppNPS.WebServer.FormLiterals.Values['Dept'] <> '' then begin
             if srchkeys = 0 then begin
                sql.add(' WHERE Dept = ');
                sql.add('''' + WebAppNPS.WebServer.FormLiterals.Values['Dept']+'''');
                inc(srchkeys);
             end
             else
                sql.add( ' and Dept = ' + '''' + (WebAppNPS.WebServer.FormLiterals.Values['Dept'])+
'''');
          end;
          if (WebAppNPS.WebServer.FormLiterals.Values['emptype'] <> 'ALL') and
              not (WebAppNPS.WebServer.FormLiterals.Values['emptype'] = '')  then begin
             if srchkeys = 0 then begin
                sql.add(' WHERE Status = ');
                sql.add('''' + WebAppNPS.WebServer.FormLiterals.Values['emptype'] + '''');
             end
             else
                sql.add( ' and Status = ' + '''' +
WebAppNPS.WebServer.FormLiterals.Values['emptype']+ '''');
          end;
          sql.add( ' ORDER BY Last, First');
      end;
      memo1.text := query1.text;
end;

procedure TWebAppNPS.NPS_ANSWERSection(Sender: TObject; Section: Integer;
  var Chunk, Options: string);
begin


  if section>1 then exit;

  WebAppNPS.Literal[ 'wgrid1.DisplaySet' ]:= 'normal';
  if WebAppNPS.Command=cCheckboxes then begin
    {Reset the query when we arrive on this page, coming from homepage}
    {Leave it alone if command is something like grid.next }
    Query1.close;
    Query1.open;
  end;
  if (length(WebAppNPS.Command) = 1) and (WebAppNPS.Command[1] in ['A'..'Z']) then begin
     WebAppNPS.WebServer.FormLiterals.Values['Last'] := WebAppNPS.Command;
     Query1.close;
     Query1.open;
  end;
  {get the currect db for the current surfer}
  SetSurferDb(SurferTbl);

end;

procedure TWebAppNPS.WebAppNPSEventMacro(Sender: TWebOutputApp;
  const aMacro, aParams, aID: string);
var
  fld:TField;
  lMacro:String;
  s:String;
begin
  lMacro:=lowercase(aMacro);
  { create the meaning of the getfield macro }
  { usage: getfield|fieldname -- see mypages.htm }
  with WebAppNPS.WebOutput do begin
    if lMacro = 'getfield' then begin
      with SurferTbl do begin
        fld:=findField(aParams);  { aParams is info after vertical bar, i.e. fieldname }
        if fld<>nil then begin
          s:=fld.asString;
```

```
          Send(s);
          end
        else
          SendComment( 'Invalid fieldname: ' + aParams );
        end;
      end
    else if lMacro = 'getfieldreg' then begin
      with registerTbl do begin
        fld:=findField(aParams);  { aParams is info after vertical bar, i.e. fieldname }
        if fld<>nil then begin
            if (fld.asString='') and (comparetext(aParams,'picture')=0) then
               Send('nopic.gif')
            else  begin
               s:=fld.asString;
               Send(s);
            end;
        end
        else
          SendComment( 'Invalid fieldname: ' + aParams );
        end;
      end
    else if lMacro = 'getfieldmain' then begin
      with main do begin
        fld:=findField(aParams);  { aParams is info after vertical bar, i.e. fieldname }
        if fld<>nil then begin
          s:=fld.asString;
          Send(s);
          end
        else
          SendComment( 'Invalid fieldname: ' + aParams );
        end;
      end
    else begin
      SendComment( 'Invalid custom macro event: ' + aMacro );
      end;
    end;
end;


procedure TWebAppNPS.NPS_DETAILSection(Sender: TObject; Section: Integer;
  var Chunk, Options: string);
var
  s,user,picfilename:string[128];
begin

  if section > 1 then exit;
  {our job here is just to locate to the correct record }

  s := webcommandline1.mcommand;
  SetSurferDb(SurferTbl);
  SurferTbl.locate('Id',s,[]);
  {the surfer table should include the registertbl (i.e., the query should execute
   but this is a quick an outer join on main.db with registertbl.db.  then we can
   get rid of the getfieldreg macro (I think?)...}
  if not(Registertbl.locate('FkID',s,[])) then
     Registertbl.locate('FkID','blank',[]);     // this is a temp fix for those
                                                // who are not registered
                                                // it points to a blank record
                                                // in the register tbl
end;

procedure TWebAppNPS.SetSurferDb(Tbl:TTable);
begin

   if Tbl.tablename = WebAppNPS.Literal['DBName'] then exit;

   Tbl.close;
   Tbl.tablename := WebAppNPS.Literal['DBName'];
   Tbl.open;
   SetDBEnv(Tbl);
end;
```

```
procedure TWebAppNPS.SetDBEnv(Tbl:TTable);

begin
  with Tbl do begin

    fieldByName('Last').displayLabel:='LAST NAME';
    fieldByName('First').displayLabel:='FIRST NAME';
    fieldByName('Email').displayLabel:='<IMG SRC=%=imageDir=%mailto.gif> EMAIL';
    fieldByName('Phone').displayLabel:='<IMG SRC=%=imageDir=%phone1.gif>PHONE';
    fieldByName('Middle').displayLabel:='MI';
    fieldByName('URL').displayLabel:='HOMEPAGE (URL)';

    {set hot field}
    fieldByName('Last').tag:=HotFieldTag;
    fieldByName('Email').tag:=HotFieldTag;
    fieldByName('URL').tag:=HotFieldTag;
  end;
end;

procedure TWebAppNPS.NPS_ANSWERREGSection(Sender: TObject;
  Section: Integer; var Chunk, Options: string);
begin
  if section>1 then exit;
  WebAppNPS.Literal[ 'wgrid2.DisplaySet' ]:= 'normal';
  if WebAppNPS.Command=cCheckboxes then begin
    {Reset the query when we arrive on this page, coming from homepage}
    {Leave it alone if command is something like grid.next }
    Query1.close;
    Query1.open;
  end;
  {get the currect db for the current surfer}
  SetSurferDb(SurferTbl);
end;

procedure TWebAppNPS.NPS_DETAILREGSection(Sender: TObject;
  Section: Integer; var Chunk, Options: string);
var
  s:string[128];
begin
if section>1 then exit;

// I need this here so that the conditional will work properly
WebAppNPS.Literal[ 'FailRegister' ]:= '';
  {our job here is just to locate to the correct record }

//****************************************
//*  VERY IMPORTANT!!  THIS LATER NEEDS TO BE THE MAIN DATABASE ON THE DEC...NOT
//*  THE TEMP QRY*.DB TABLES!!!!  SAME GOES FOR THE OTHER DETAIL "ONSECTION"
//*  PROCEDURE!!!!!!!!!
//****************************************

  s := webcommandline1.mcommand;    // we are using webCommandLine1 instead of Webapp.command
  WebAppNPS.Literal['MainID'] := s; // because  the webapp.command translates to uppercase
  SetSurferDb(SurferTbl);
  with SurferTbl do begin
     locate('Id',s,[]);
  end;
end;

procedure TWebAppNPS.NPS_FBREGISTERExecute(Sender: TObject;
  var Continue: Boolean);
var
  surferEMail,password:string;
begin
  surferEMail:=WebAppNPS.Literal['emailreg'];
  if surferEMail='' then begin
    with WebAppNPS do begin
      SendString( 'You have to enter your email address!<p>' );
      continue:=False;
```

```
    end;
  end
  else
     with registertbl do begin
        open;
        WebAppNPS.Literal[ 'FailRegister' ]:= '';
        if findkey([surferEmail]) then begin
           WebAppNPS.Literal[ 'FailRegister' ]:= 'T';
           BounceTo('DetailRegFail');
        end
        else  begin
           password := generatePassword;
  //****  Should use the dynamic creation of a query and then use SQL Insert because
  //      another surfer could change the record pointer...leaving you with update
  //****  anomalies ... this really shouldn't be a problem if we are appending..
  //      right???
           append;
           fieldbyname('Password').asString := password;
           fieldbyname('ID').asString := trim(surferEmail);
           fieldbyname('FkID').asString := WebAppNPS.Literal['MainID'];
           fieldbyname('activated').value := false;
           post;

           // What if mail fails???  should I use expiration date...or some kind of
           // exception on email failure????
           with webmail1 do begin
              Mailto.clear;
              Mailto.add(surferEmail);
              Lines.clear;
              Lines.add('Once you login your account will be activated');
              Lines.add('Your Login is:  ' + surferEmail);
              Lines.add('Your password is:  ' + password);
              subject := 'Registration Information';
              execute;
           end;
        end;
     end;

end;


//  This is not a sophisticated password generator...it is used only to demonstrate
//  the concept within this application's environment and it serves the purpose.
function TWebAppNPS.GeneratePassword :string;
var
   I: Integer;
   Pass: string;
begin
   Randomize;
   Pass:= '';
   for I := 1 to 6 do begin
     Pass := Pass + chr(Random(26)+65);
   end;
   generatePassword := Pass;
 end;

procedure TWebAppNPS.NPS_DETAILLOGSection(Sender: TObject; Section: Integer;
  var Chunk, Options: string);
var   s:string;
begin
   if section > 1 then exit;
// probably should use a unique query here...another surfer could change the
// position of the record pointer.  You should create a query on the fly??? and
// either assing values to literals and "free" the query... or ..."free" the Query
// using the "onexecdone" event?!  (Remember...you will also have to change the
// getfields macros...eeee can we do this???)  Maybe we don't have to use getfields
// we can just use the literals...then use a query to update....

   s := registertbl.fieldbyname('FkID').asstring;
   main.locate('ID',s,[]);
```

```
end;

procedure TWebAppNPS.NPS_DETAILLOGExecute(Sender: TObject;
  var Continue: Boolean);
var s:string;
begin

memo1.text := 'lok= ' + WebAppNPS.Literal['LoginOk'] + ' loginPend= ' +
    WebAppNPS.Literal['LoginPending'] + ' getlit(login)= ' + getliteral('login');

  if not( WebAppNPS.Literal['LoginOk'] = 'TRUE') and
       not(WebAppNPS.Literal['LoginPending'] = 'TRUE') then begin
     BounceTo('LoginPg');
     exit;
  end;
  if not(WebAppNPS.Literal['LoginPending'] = 'TRUE') then
   exit;

  {We only want to do this if the user is trying to login}
  s:= getLiteral('Login');
  if not(registerTbl.locate('ID',s,[locaseinsensitive])) then begin
     BounceTo('LoginFail');
  end
  else  if registerTbl.fieldbyname('password').asstring <> WebAppNPS.Literal['password'] then begin
          BounceTo('LoginFail');
        end
        else
          WebAppNPS.Literal['LoginOk'] := 'TRUE';
  WebAppNPS.Literal['LoginPending'] := 'FALSE';
end;



procedure TWebAppNPS.NPS_FBUPDATEExecute(Sender: TObject;
  var Continue: Boolean);
var updateQry:TQuery;
begin
  if WebAppNPS.Literal['LoginOk'] <> 'TRUE' then begin
     BounceTo('Login');
     exit;
  end;
  updateQry := Tquery.Create(Self);
  with updateQry do begin                //************************************
     databasename := 'locator';  // make this a constant so we only have
                                 //  to change once.
     //updatemode := [wherekeyonly];
     sql.add('update REGISTER ');
     sql.add('set URL2 = ''' + getFormLiteral('url2') + ''',');
     sql.add('    PHONE2 = ''' + getFormLiteral('phone2') + ''',' );
     sql.add('    FAX2 = ''' + getFormLiteral('fax2') + ''',' );
     sql.add('    COMMENTS = ''' + getFormLiteral('comments') + ''',' );
     sql.add('    ACTIVATED = ''1'' ' );
     sql.add('where ID = ''' + getLiteral('login') + '''');

memo1.text := sql.text;
     execsql;
     close;
     sql.clear;
     sql.add(' update MAIN m set m.registered = ''1'' ');
     sql.add(' where m.id = (select FkID from REGISTER ');
     sql.add('          where ID = ''' + getLiteral('login')  + ''')');
memo1.text := sql.text;
     execsql;
     free;

  end;
end;

procedure TWebAppNPS.NPS_HOMEPAGEExecute(Sender: TObject;
  var Continue: Boolean);
begin
```

```
   if (WebAppNPS.Command = 'LOGOUT') then begin
      WebAppNPS.Literal['LoginOk'] := 'FALSE';
      WebAppNPS.Literal['LoginPending'] := 'FALSE';
      edit1.text := edit1.text + 'logout';
   end;
end;

procedure TWebAppNPS.NPS_LOGINPGExecDone(Sender: TObject);
begin
  WebAppNPS.Literal['LoginPending'] := 'TRUE';
end;

procedure TWebAppNPS.NPS_HOMEPAGESection(Sender: TObject; Section: Integer;
  var Chunk, Options: string);
begin
    if section > 1 then
      exit;
    if (WebAppNPS.Command = 'LOGOUT') then begin
      WebAppNPS.Literal['LoginOk'] := 'FALSE';
      WebAppNPS.Literal['LoginPending'] := 'FALSE';
      edit1.text := edit1.text + 'logout';
    end;
end;

{Utility Functions}
function TWebAppNPS.GetLiteral(LiteralName :string):string;
{Read a literal                                                    }
begin
     result := WebAppNPS.Literal[LiteralName];
end;
{-------------------------------------------------------------}

function TWebAppNPS.GetFormLiteral(LiteralName:string): string;
{read a form literal}
begin
     result := WebAppNPS.WebServer.FormLiterals.Values[LiteralName];
end;

Procedure TWebAppNPS.PutLiteral(LiteralName,LiteralValue:string);
{Add or update a WebApp literal}
begin
   WebAppNPS.Literal[LiteralName] := LiteralValue;
end;

procedure TWebAppNPS.BounceTo(PageID:string);
begin
   webappoutput.sendbounceto(webappnps.cgicaller + '?nps:' + PageID + ':' +
                             inttostr(webappnps.session));
end;


{********************************* TESTING AREA FOR UPLOAD ****************}



procedure TWebAppNPS.UpLoadUserFiles;
{looks for the source and companion files for this job          }
var
   uplInfo, ThisEntry : string;
   NewFileName, ClientFileName: string;
   JobFileType : string;

begin {save uploaded files}

      {We don't want to do anything if we didn't get here from the send page}

      {loop over the list of files to upload                                 }

   with webserver.webini do begin
```

```
      ThisEntry := 'uploadfile';
      section := 'Form File';
      entry := ThisEntry;
      uplInfo := StringValue;
   end;
memo1.text := uplinfo;

// Should use a constant (or ini file) for webserver path!!
   NewFileName := cImageDir + getliteral('Login');

   if uplInfo <> '' then begin
      SaveUploadFile(uplInfo,NewFileName,ClientFileName);
   end
   else begin
      WebAppNPS.WebOutput.Send
         ( '<center> <font size=5><b> No File has been' +
         ' uploaded -- you must enter a valid gif or jpeg picture file'
           + '</b></font></center><br>' )
   end;

end;

procedure TWebAppNPS.SaveUploadFile(const UploadInfo:string;
   var NewFileName,ClientFileName:String);
const MAXLEN=255;
var
  Name, EMail, Desc : string;
  UploadFileName, ServerFileName: string;
  FileSize,FilesType,FilesTypeTest:string;
  ws,wsl,wsr : string;
  Entry,StringValue:string;
  FileToDelete :array[0..255] of char;
  UploadFile: file;
  PicOK:boolean;
  s:string;

begin
   {This code relies heavily upon some very handy string
    processing a parsing routines in the ucstring.pas unit
    of the complete TPack code }

    {The following comment contains a sample value for UploadInfo:}
   {
   [C:\WebSite\cgi-temp\66ws.001] 46 text/plain binary [C:\PROJECT\README.TXT]
   }
 memo1.text := memo1.text + ' ' +UploadInfo;
   ws := UploadInfo;
   SplitString(ws,']',wsl,ws);
   ServerFileName :=RightOf('[',wsl);
memo1.text := memo1.text + ' ' + serverFilename;
   SplitString(ws,'[',wsl,wsr);
   ClientFileName := LeftOf(']',wsr);

   wsl:= TrimLeft(wsl);
   SplitString(wsl,' ',FileSize,FilesType);

   FilesTypeTest := trim(LeftOf(' ',FilesType));
   PicOK := False;

   if (FilesTypeTest = 'image/gif') then begin
      newfilename := newfilename + '.gif';
      PicOK := TRUE;
   end
   else if (FilesTypeTest = 'image/jpeg') then begin
      newfilename := newfilename + '.jpg';
      PicOK := TRUE;
   end;
   if PicOK then begin
      ForceFileRename(serverFilename,NewFileName);  //should have error checking here
      WebAppNPS.WebOutput.Send
         ( '<center> <font size=5><b> File=' + ClientFileName + '<br> has been '+
```

```
         'successfuly uploaded and saved as ' + ExtractFilename(newfilename)
            + '</b></font></center><br>' );
      s:= getliteral('Login');
      with registertbl do begin
         locate('ID',s,[]);
         edit;
         Fieldbyname('picture').asstring:= ExtractFilename(newfilename);
         post;
      end;

   end
   else
      WebAppNPS.WebOutput.Send
         ( '<center> <font size=5><b> File=' + ClientFileName + '<br> has not been uploaded because
'+
         'it was not a gif or jpg </b></font></center><br>' );

end;



procedure TWebAppNPS.NPS_FBUPLOADPICExecute(Sender: TObject;
  var Continue: Boolean);
begin
   if WebAppNPS.Literal['LoginOk'] <> 'TRUE' then begin
     BounceTo('Login');
     exit;
   end;
   uploaduserfiles;
end;

procedure TWebAppNPS.NPS_UPLOADPICExecute(Sender: TObject;
  var Continue: Boolean);
begin
   if WebAppNPS.Literal['LoginOk'] <> 'TRUE' then begin
      BounceTo('Login');
      exit;
   end;
end;

procedure TWebAppNPS.NPS_LOGUPDATESection(Sender: TObject;
  Section: Integer; var Chunk, Options: string);
  var s:string[128];
begin
      if section > 1 then exit;
// probably should use a unique query here...another surfer could change the
// position of the record pointer.  You should create a query on the fly??? and
// either passing values to literals and "free" the query... or ..."free" the Query
// using the "onexecdone" event?!  (Remember...you will also have to change the
// getfields macros...eeee can we do this???)  Maybe we don't have to use getfields
// we can just use the literals...then use a query to update....
      s:= getLiteral('Login');
      registertbl.locate('ID',s,[]);
      s := registertbl.fieldbyname('FkID').asstring;
      main.locate('ID',s,[]);

end;

procedure TWebAppNPS.NPS_LOGUPDATEExecute(Sender: TObject;
  var Continue: Boolean);
begin
   if WebAppNPS.Literal['LoginOk'] <> 'TRUE' then begin
     BounceTo('Login');
     exit;
   end;
end;


procedure TWebAppNPS.ForceFileRename(const oldName,newName:string);
begin
   if DeleteFile(pchar(newName))then      //I shouldn't have to type cast as PChar!!!
```

```
      edit1.text := 'delete';
   if RenameFile(oldName,newName) then
      edit1.text := edit1.text + 'rename';

end;
procedure TWebAppNPS.NPS_FBUPLOADPICSection(Sender: TObject;
  Section: Integer; var Chunk, Options: string);
var s:string;
begin
   if section > 1 then exit;
   s:=getliteral('login');
   registertbl.locate('id',s,[]);
end;

procedure TWebAppNPS.NPS_CHGPASSExecute(Sender: TObject;
  var Continue: Boolean);
begin
   if WebAppNPS.Literal['LoginOk'] <> 'TRUE' then begin
      BounceTo('Login');
      exit;
   end;
end;

procedure TWebAppNPS.NPS_FBCHGPASSExecute(Sender: TObject;
  var Continue: Boolean);
  var s:string;
begin
   if WebAppNPS.Literal['LoginOk'] <> 'TRUE' then begin
      BounceTo('Login');
      exit;
   end;
   if getFormLiteral('chpass1') <> getFormLiteral('chpass2') then
      BounceTo('ChgPassFail')
   else begin

      s:= getLiteral('Login');
      edit1.text := s + ' ' + getFormLiteral('chpass1');
      with registertbl do begin
         locate('Id',s,[]);
         edit;
         fieldByName('password').asString := getFormLiteral('chpass1');
         post;
      end;
   end;
end;


end.
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
   8725 John J. Kingman Road, Suite 0944
   Fort Belvoir, VA 22060-6218

2. Dudley Knox Library.....2
   Naval Postgraduate School
   Dyer Road
   Monterey, CA 93943-5101

3. Director, Training and Education.....1
   MCCDC, Code C46
   1019 Elliot Road
   Quantico, VA 22134-5027

4. Director, Marine Corps Research Center.....2
   MCCDC, Code C40RC
   2040 Broadway Street
   Quantico, VA 22134-5107

5. Director, Studies and Analysis Division.....1
   MCCDC, Code C45
   3300 Russell Road
   Quantico, VA 22134-5130

6. Marine Corps Representative.....1
   Naval Postgraduate School
   Code 037, Bldg. 234, HA-220
   699 Dyer Road
   Monterey, CA 93940

7. Marine Corps Tactical Systems Support Activity.....1
   Technical Advisory Branch
   Attn: Maj. J.C. Cummiskey
   Box 555171
   Camp Pendleton, CA 92055-5080

8. Prof. James C. Emery (Code 05).....1
   Naval Postgraduate School
   Monterey, CA 93943-5103

9. Prof. C. Thomas Wu (Code CS/Wq).....1
   Naval Postgraduate School
   Monterey, CA 93943-5103

10. Jeffrey E. Forte.....1
    5913 Dungeness Lane
    Alexandria, VA 22315