1. ABSTRACT

This project creates e-Plan system for Chemical Plants in the Corpus Christi area. The e-Plan system is designed to act as a tool for quick and efficient response in the event of the release of a hazardous material. This e-Plan system consists of a database to maintain records of plant information such as facilities, building information, chemical inventory, processes, vulnerability analysis, emergency plans, employee training, and site clean-up contractors. It also provides a user-friendly interface for generic database manipulations via the Internet. This e-Plan system will allow authorized users to generate the various reports for the EPA, Coastal Guard, Department of Transportation, Department of the Interior and Department of Labor.
# TABLE OF CONTENTS

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

TABLE OF CONTENTS ........................................................................................................ iii

LIST OF FIGURES ................................................................................................................. iv

1. BACKGROUND AND RATIONALE ............................................................................... 1

2. NARRATIVE ................................................................................................................... 5

   2.1 Overview ................................................................................................................... 5
   2.2 Function of the Web Based ICP Database System .................................................... 7
       2.2.1 Security Page ...................................................................................................... 8
       2.2.2 Main Page .......................................................................................................... 9
       2.2.3 Database Manipulation Page ............................................................................. 9
       2.2.4 Search Page ..................................................................................................... 11
       2.2.5 Reports Generation Page .................................................................................. 11

3. ENVIRONMENT ............................................................................................................. 12

4. PROCEDURE ................................................................................................................ 13

   4.1 Client Requirement and Analysis .......................................................................... 14
   4.2 Installation of Linux and Apache ............................................................................ 16
   4.3 Installation and Configuration of LDAP .................................................................. 16
       4.3.1 Introduction of LDAP ...................................................................................... 16
       4.3.2 Developing Directory Information Tree .......................................................... 20
   4.4 Interface Script Development ............................................................................... 25
   4.5 Testing e-Plan System ............................................................................................ 28

5. EXPECTED RESULTS .................................................................................................. 29

6. REFERENCE ................................................................................................................ 30

7. PROGRAM LISTINGS .................................................................................................. 31
LIST OF FIGURE

FIGURE 1. ICP Data Flow Diagram Level 1 .................................................. 6

FIGURE 2. System Overview................................................................. 15

FIGURE 3. LDAP Directory Tree ......................................................... 19
1. BACKGROUND AND RATIONALE

Corpus Christi is located on the southeast coast of the Gulf of Mexico. It is ecologically and economically important because of biological diversity and intensive recreational and fishing activities. Many petrochemical industries came to this area with the economic development of the city, thus the Corpus Christi area suffers frequent oil and hazardous material releases, which often result in environmental damage and threaten nearby communities. Between the years of 1987 and 1997 there were a total of 742 oil or hazardous material releases, of which seven caused injuries and eight required evacuations.

Due to these conditions, facility operators normally face a myriad of federal, state and local emergency-response planning requirements. The requirements originating from the Oil Pollution Act (OPA-90), Clean Water Act, Resource Conservation and Recovery Act (RCRA), Clean Air Act, and Occupational Safety and Health Act (OSHA) can lead to as many as nine separate facility response plans. The traditional approach to satisfying these requirements has been to develop separate plans for each set of regulations. Many of the individual plan elements are similar, if not duplicative, while response planning requirements vary with respect to technical considerations and the degree of detail required. The inefficiency of these requirements is most clear for operators of facilities.
The National Response Team offers an efficient format called the “One Plan” to combine planning criteria into one effective document that is readily accepted by regulatory authorities. The primary benefit of the One Plan is administrative cost saving. With review, revision, updating, and reproduction costs approaching $100.00 per page, facility planners have found that the integrated response plan makes a facility better prepared to respond. An integrated approach to response planning brings responsible managers and planners together in the development of a plan that focuses on common procedures and a facility-wide response management framework.

The Environmental Protection Agency (EPA) and local emergency planning committees are developing an electronic integrated contingency plan called “e-Plan” to help local industries consolidate information on chemical hazards, response procedures, and response resources. This e-Plan will reduce the existence of multiple, uncoordinated contingency plans at a facility to facilitate response actions. The e-Plan will benefit personnel from all of the organizations and these chemical facilities and will serve as a tool for risk communication to the surrounding community.

The e-Plan project group consists of the EPA Region VI, the National Spill Control School and the Texas A&M University – Corpus Christi (TAMUCC) Department of Computing and Mathematical Sciences. The ICP project began with the Microsoft Access-based BOLDER (Basic On-Line Disaster Emergency Response) planning tool, which is a software planning tool originated by the Chandler Fire Department. The BOLDER program can develop a single emergency response plan using the One-Plan
format developed by the National Response Team. As a test of BOLDER’s capabilities, the Corpus Christi group distributed the BOLDER packages to the following volunteer facilities:

Celanese Chemicals Division
Equistar Chemicals
City of Corpus Christi Water Department
Valero Refining Company
Formosa Plastics
Koch Refining Co., LP
US Naval Station Ingleside
H. E. Butt Grocery Co.
Citgo Refining
Elementis Chromium
Refinery Terminal Fire Company
Corpus Christi Area Oil Spill Control Association
Texas General Land Office - Oil Spill Division
Texas Natural Resources Conservation Commission
US Coast Guard - MSO Corpus Christi
Local Emergency Planning Committee- Nueces County
Corpus Christi Fire Department

After using Microsoft Access-based BOLDER, most of the users complained about the inconvenience of distributing and moving around data between different computers. One reason is that the database requires more than 1.44 MB in memory and can not be loaded onto a standard floppy disk, and most of the users do not have a CD writer or Zip drive connected to their computer. The EPA also would like to see the One-Plan done through the Internet. So the expansion of the One-Plan to the World Wide Web becomes necessary and important for the further development of the One-Plan so that all of the users will be able to get the most updated information efficiently on time.

This project fits perfectly in the environmental engineering application area of my Computer Science Master’s Degree. The project involves many of the chemical
emergency plans required by environmental regulation, which were covered in my
previous Environmental Engineering study. Of course, the design and implementation of
this web-based database system requires a strong computer-science background. With my
past degrees in Environmental Engineering and current knowledge of computer science, I
am well suited to carry out these tasks.

This e-Plan project involves designing and developing a front-end data entry tool and
report-generating tool on the World Wide Web with an OpenLDAP directory server
system as the backend database. The e-Plan allows the chemical facilities to insert, delete,
and update information on emergency notification and response procedures, response
management systems, response resources, and hazards present at facilities. The e-Plan
also prevents other unauthorized users from modifying and accessing the facility-
confidential information through the Internet. Some of the information is available to the
community. This e-Plan serves the purpose of generating various reports required by
different organizations through the Internet, which offers flexibility and cost-reductions
to chemical facilities. This e-Plan uses basic computer science concepts like
programming, data management system, networking, and Web security. The e-Plan is an
efficient and cost-effective alternative to the current manual system.
2 NARRATIVE

This section describes the features of the e-Plan Database System as they appear to the user.

2.1 Overview

The primary goal of the e-Plan is to provide a report-generating tool on the World Wide Web for chemical storage facilities. This user-friendly, interactive tool provides immediate up-to-date information about facilities to its users and generates the reports for the requests from various organization and government agencies.

The following data flow diagram in Figure 1 represents an overview of the e-Plan data flow:
Figure 1. e-Plan Data Flow Diagram Level 1

The chemical storage facilities and plants participating in this project (1) gather the information necessary to meet the e-Plan requirement (2). The facilities then connect to the e-Plan system and provide a username/password to gain access (3). Once authenticated, the facility operator fills out Web-based forms to enter the data into the database (4). This information is stored into LDAP server (5). The facility can then use other web pages on the server to generate an e-Plan report for submission to an agency (6), and emergency responders can quickly access the database in the event of an emergency (7). The facility information is stored in the database and can be updated for later development. Also, city fire department, corresponding facilities and departments, or even community officials can fetch information during hazardous chemical emergency.
2.2 Function of the Web-based ICP database system

Specifically, after the e-Plan system authenticates the user, the user is able to:

- add new records to the database for his/her facility,
- update and delete records for the facility,
- query the database, e.g., for the location of chemicals,
- generate and print regulatory reports,
- grant privileges to other users to view specific information,
- provide regulatory data to agencies and similar organizations, and
- publish general information of use to the community.

Access to the data through the World Wide Web utilizes 5 main sections: a security page, a main menu page, a search page, a report generation page, and a data entry/update page.

Each of these sections is described below:

2.2.1 Security Page

In general, the e-Plan divides users into four groups.

i. Facilities – chemical storage facilities and plants which use e-Plan to enter/update plans.

ii. Government agencies and organizations – Environmental Protection Agency, and other similar organizations.

iii. Responders – City Fire Department, and

iv. Guests – general user such as local community, and research groups.

Whenever a user attempts to enter the URL of any page in e-Plan for the first time, he or she is directed to a security check page, where he/she is prompted to enter his/her
distinguished name (DN) and password. At present, this security is necessary to avoid any database manipulation by any unauthorized user. The e-Plan administrator assigns the DNs and passwords to the certified users who join this project. Later, a facility can grant or restrict other regulatory agencies, organizations, and facilities the ability to view particular information about itself. Facilities can thus prevent competitors from viewing each other's confidential data. Community users are allowed to “Guest” login and view the non-confidential parts of the system.

2.2.2 Main Page

After a user passes the security page, he/she will be sent to a main menu specific to the type of user. Each type of main menu guides the user through the necessary steps to perform different tasks. A typical menu has the following options:

i. Add new facility,

ii. Retrieve and update existing facility,

iii. Search for facility information,

iv. Generate reports, and

v. Specify the contents that can be viewed by government and regulatory agencies, community, and other facility users.

According to the user's privilege, he/she is only allowed to do certain operations. For example, if a user is classified as the City Fire Department, he/she will be able to view most of the information granted by different facilities. Otherwise, if the user is a guest, he/she is able to view only some general information.
2.2.3 Database Manipulation Pages

When a facility user chooses either of the first two options in the main pages, he/she will be directed to these pages. Each database manipulation page has different forms for each type of information to be stored.

Facilities Section: This is the basic facility information entered before entering any other type of information. Facilities can enter more than one facility into the e-Plan. Data in this section include general facility information, facility owner and property owner, technical and public contacts, emergency contacts, and facility site plans.

Chemical Inventory Section: This holds the characteristics and storage information of different chemicals, compounds, and products, which might be involved in an incident. Users can enter data into the chemical type, exposure routes, transportation methods, and attached Material Safety Data Sheets (MSDS) information.

Vulnerability Analysis Section: This describes each of the types of hazards and the population, critical facilities, and environmental impact that may be affected.

Emergency Planning Section: This describes all considerations covered by a specific standard operating procedure. This section also documents the “chain of command”: who is the qualified individual in a given scenario, where to contact
specific individuals and regulatory authorities. It addresses worker health
assessment and personal protective equipment needed in or around the facility.

Employee Training Section: This contains the different areas in which employees are
trained in different procedures and practices. Users enter information in this form
disclosing which employees are involved and to whom they are to report. In the
case of emergency, personnel information may be retrieved.

Site Clean-up Contractor Section: This section tracks all the contractors the facility can
call for hazardous waste and site clean-up. Users enter data on clean-up
techniques, disposal alternatives, and HAZMAT team information including
addresses and 24-hour contact phone numbers.

2.2.4 Search Page

This page gives people the ability to search the e-Plan database by facility name, which
are listed in a combo box or users are able to fill it in the text field. After a new facility is
entered, the facility names will be updated in the combo box. Users are able to retrieve a
set of attributes according to the object class from these pages by choosing the facility
name.

2.2.5 Reports Generation Page

This page is used to generate different reports required by different regulatory agencies.
Currently, the goal of this project is to develop an electronic emergency response plan for
the EPA Facility Response Plan (FRP), thus reducing the potentially 500-plus pages of documents into a concise printable plan. Future extensions to e-Plan will provide data to generate other reports such as the CAA Risk Management Plan (RMP) Submit, and the National Response Team (NRT) “One Plan”.
3. PROJECT ENVIRONMENT

The e-Plan can be displayed through any Web browser such as Netscape or Internet Explorer. Furthermore, provisions are made for users still using older version browsers like Netscape 3.0. The database has been developed using OpenLDAP system by Lightweight Directory Access Protocol under RedHat Linux 6.1 on a PC. Apache is used as the Web Server software. The Hypertext Markup Language (HTML) is used to generate the forms, and Common Gateway Interface (CGI) scripts are used to access the LDAP database and to dynamically generate the HTML forms. The Perl module Net::LDAP is used to interface with the database.
4. PROCEDURE

The e-Plan described in section 3 of this proposal has been implemented according to the following steps:

1. Meet with clients including EPA contractor, environmental consultant companies, and industry representatives. Understand the users’ requirements and the system security requirements.

2. Choose components needed to implement the e-Plan. As of this writing, the e-Plan is using the Linux RedHat 6.1 version operating system, the Apache Web server, and the OpenLDAP (Lightweight Directory Access Protocol) directory server.

3. Install and configure Linux in one Intel i386 PC Apache 1.3.9 from ftp://ftp.apache.org/dist and the OpenLDAP distribution from RedHat.

4. Reconfigure the Apache Web Server for use with e-Plan. Read LDAP manual written by University of Michigan to manipulate the LDAP directory server and database system. Read the book Implementing LDAP.

5. Choose a suitable language – Perl for this system. Install Net::LDAP module in the system from the Comprehensive Perl Archive Network (CPAN). Order three useful books: Programming Perl for DBI, Programming Perl, and Perl Cookbook.

6. Develop and modify the database needed for the e-Plan in the LDAP directory server.

7. Add several sample data sets from the facilities into the LDAP directories created in the above process.
8. Implement the CGI/Perl programs to generate forms and provide the interactivity between users and server.

9. Optimize the graphical user interface in the Internet.

10. Test the Web-based e-Plan and write some online help routines for the users.

4.1 Client Requirements Analysis

Since the e-Plan is posted on the Internet, and multiple users are able to access it from anywhere in the world, the security, reliability, and multi-processing capabilities are critical factors in this project. Most facility users would like to have full control of the information in the database, and they should also be able to grant the different access levels to other users. Users such as fire departments need the option to quickly search the database from the browser when an emergency happens. Based on the requirements for the project, we have made the following selections:

- We chose the RedHat Linux operating system as the operating system. Linux offers one of the most powerful and reliable systems available, and as an open-source system, it can be altered to meet the needs of its users.

- We have chosen Apache HTTP Server as our Web server. Apache is a robust, commercial-grade, featureful, and freely-available source code implementation of an HTTP (Web) server, which is perfectly fit to our requirements.

- To enhance the security and efficiency of this project, we choose open LDAP (Lightweight Directory Access Protocol) directory and database server. Traditional databases are not sufficiently for this project. Furthermore, the LDAP can contain more descriptive, attribute-based information and give quick-response to high-volume
lookup or search operations for this project. LDAP is easier to manage and maintain, provides attribute-level security features, and can be distributed across multiple servers.

- Perl modules provide two services in this project. One is Net::LDAP, which allows connections to the directory server and makes queries including add, delete, update and search. The other is to write CGI applications for the web, creating HTML forms and parsing their contents for this project. Perl provides a simple interface for parsing and interpreting the query strings passed to CGI scripts, and it also offers a rich set of functions for creating fill-out forms.

Figure 2 is a simplification of how the e-Plan system works: Communication between the user’s browser and Apache is performed using HTTP and HTML. Apache executes Perl scripts using CGI, and the Perl scripts communicate with the OpenLDAP server by using Net::LDAP.

![Diagram](image)

**Figure 2. e-Plan System Overview**
4.2 Installation of Linux

The Red Hat Linux 6.1 operating system incorporates easy installation, software update information and access, and improved system management capabilities. These features enable users to harness the power of open-source software in the demanding enterprise and Internet applications that run the day-to-day business of companies worldwide.

4.3 Configuration of Apache Web Server

Apache 1.3.9 was downloaded and installed to /usr/etc/httpd. Some of the default configuration directives were changed in Apache's configuration file /etc/httpd.conf (Appendix A). Also, a ScriptAlias directive was added to httpd.conf to allow CGI scripts to be executed from the server side in the directory /home/httpd/cgi-bin.

4.4 Installation and Configuration of LDAP

4.4.1 Introduction to LDAP

LDAP is the Lightweight Directory Access Protocol, which is a directory service protocol that runs over TCP/IP. It is a type of specialized database application which allows a user to ask the database for some information (called a query) and returns all the data that matches that query in the form of records. LDAP is a protocol that defines how a particular kind of data, directory data, should be structured for easy retrieval in a global scope using the TCP/IP. Several reasons lead to the choice of LDAP as the database model for this ICP project:
1. LDAP, as an Internet-based protocol, is designed to run on any hardware, operating system and language that can run the basic Internet protocol TCP/IP. It allows multiple facility users and government agencies to use the database server.

2. The directory server doesn’t usually implement a complicated transaction, however it gives quick response to high-volume lookup or search operations, which is of critical importance to emergency responders.

3. LDAP has built-in facilities to protect the e-Plan system data from unauthorized access. By using LDAP, facility users can control access to their data protect the information the server contains.

4. The LDAP directory service model is based on entries. An entry is a collection of attributes that has a name, called a distinguished name. Each of the entry’s attributes has a type and one or more values. The values depend on what type of attribute it is. So in this project, traditional relational database tables become unnecessary. Instead of tables, the e-Plan system creates several entries and each entry has a set of attributes. Facility users are able to update these attributes easily according to the requirements from the government agencies.

5. Since many components of e-Plan are still under development, it is difficult to specify a database format right from the start. In a relational database management system, one must define tables with all attributes known at design time. However, LDAP allows attributes to be added dynamically, and entries are allowed to have different sets of attributes.

6. LDAP also offers the ability to create redundant servers to duplicate information in different locations. This allows responder groups to maintain local copies of e-Plan
data for use in responding to emergencies (when the primary LDAP server may be inaccessible).

7. LDAP allows the data and queries to be distributed across multiple servers. Facilities that want to keep on-site control of their data but still make use of e-Plan can set up their own LDAP server. This facility-owned LDAP server can then be integrated into the e-Plan system to provide information to the general public and emergency responders.

In LDAP, directory entries are organized in a hierarchical structure. Figure 3 shows an example of LDAP directory tree. Each box in Figure 3 represents an LDAP entry. An entry has a distinguished name (DN) that uniquely identifies the entry and consists of the path from the entry to the root. An entry can also contain an arbitrary number of attributes of the form “name=value”. The attributes allowed and required for a given entry are determined by the entry’s objectclasses. For example, an entry of objectclass Facility might have required attributes FacilityName, Address, PostalCode, and optional attributes ParentCompany, and SICCode.
Figure 3. LDAP Directory Tree

LDAP also defines operations for interrogating and updating the directory. These operations can add and delete an entry from the directory, change an existing entry, and change the name of an entry. However, LDAP considers that updates to the directory will occur relatively infrequently and that the primary operation is search and retrieval. Therefore, LDAP has been organized to provide highly scalable search and retrieval operations. Thus LDAP is well-suited to meet the requirements for e-Plan.
4.4.3 Developing Directory Information Tree

In the LDAP server, directory data is represented as a Directory Information Tree (DIT). Instead of tables and fields the system creates several Distinguished Names (DN) entries with many attributes. Attributes are generally strings of characters, but can also be images or other binary objects.

1. Naming the root:

   The organization e-Plan is used as the root of the DIT:

   o=e-Plan, c=us

2. Determining RDNs: Relative Distinguished Names (RDNs) are the parts that comprise the DN of an attribute. In general, RDN is the left-most part of a DN. There are some possible choices for a RDN including common name, universally unique identifier, personal security number, user ID, and e-mail address. This project uses the LDAP common name attribute (cn) to identify facilities and other entities in the system.

   For example: the DN for the O.N.Stevens Water Filtration Plant might be:

   cn=O.N.Stevens Water Filtration Plant, o=e-Plan, c=us

3. Defining objectclasses: every facility in e-Plan has Facility information, Building Information, Chemical information, Processes, Emergency Plan, Employee Training, Vulnerability Analysis and Site Clean-up Contractor.

   The objectclass defined for e-Plan are: facility, information, building information, chemical information, process, vulnerability analysis, emergency plan, employee training and site cleanup contractor. Each of these are described in the sections below.
a. Facility Information.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Parent Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>FacilityName</td>
<td>FacilityUse</td>
</tr>
<tr>
<td>Address</td>
<td>Latitude</td>
</tr>
<tr>
<td>City</td>
<td>Longitude</td>
</tr>
<tr>
<td>County</td>
<td>OilStorageStartUp</td>
</tr>
<tr>
<td>State</td>
<td>SICCode</td>
</tr>
<tr>
<td>PostalCode</td>
<td>OtherIDNumbers</td>
</tr>
<tr>
<td>EmergencyContactID</td>
<td>EmergencyPhoneNumber</td>
</tr>
<tr>
<td>EmergencyName</td>
<td>EmergencyTitleInternal</td>
</tr>
<tr>
<td>LoadUnload</td>
<td>Roads</td>
</tr>
<tr>
<td>Parking Plans</td>
<td>StormSewers</td>
</tr>
</tbody>
</table>

Example:
```
dn: ou=facility, cn=O.N.Sevens Water Filtration Plant,o=e-Plan, c=us
FacilityName=O.N.Stevens Water Filtration Plant
Address=13101 Up River Road
City=Corpus Christi
County=Nueces
PostalCode=78410
ParentCompany=City of Corpus Christi
FacilityUse=Water Treatment Plant - Water Production
Latitude=27 51 38 North
Longitude=97 37 17 West
SICCode=4941
OtherIDNumbers=N/A
FacilityOwnerName=Water Department of Corpus Christi, Texas
FacilityPhoneNumber=3618611216
MailingAddress=2726 Holly Road
FacilityFaxNumber=N/A
OwnerCity=Corpus Christi
OwnerState=Texas
OwnerPostalCode=78412
PhysicalAddress=13101 Up River Road
```
b. Building Information

*Attributes:*
- BuildingName: Skylight
- HVACEquipment: CrossSectional
- ScrubberExhaust: ConstructionType
- AccessHatches: LocaleofStorage
- SmokeHatches: Identification
- QuantitiesinStorage

*Example:*
- BuildingName=Water Filtration Plant
- HVACEquipment=located in the basement of the main building
- ScrubberExhaust=located on roof of the chemical building
- AccessHatches=located on the roof of the main building
- Skylight=N/A
- CrossSectional=see roof plan
- ConstructionType=type I fire resistive
- LocaleofStorage=chemicals are stored in the Chemical Facility only.
- Identification=The drumming area is located in the chemical facility
- QuantitiesinStorage=50,000

c. Chemical Information

*Attributes:*
- ChemicalName

*Example:*
- dn: ou=chemical, cn= O.N.Stevens Water Filtration Plant, o=e-Plan, c=us
- ChemicalName=Aluminum Sulfate

d. Processes.

*Attributes:*
- DescriptionofProcess
- HazardClass
- LocationofProcess

*Example:*
- dn : ou=processes, cn= O.N.Stevens Water Filtration Plant, o=e-Plan, c=us
- DescriptionofProcess=Generation of Chlorine Dioxide
- LocationofProcess=Generated in the gas measurement
building in 750 gallon day tank
HazardClass=Reactive and EHS

e. Vulnerability Analysis
Attributes:
VULZoneAssessment MediumVULCriticalFacility
VULPopulation MediumVULEnvironmental
VULPublicReceptors WorstVULZoneAss
VULCriticalFacilities WorstVULPopulation
VULEnvironmental WorstVULReceptors
MediumVULZoneAss WorstVULCriticalFacility
MediumVULPopulation WorstVULEnvironmental
MediumVULReceptors

Example:
dn: ou=vulnerability, cn= O.N.Stevens Water Filtration Plant, o=e-Plan, c=us
VULZoneAssessment=.75 mile radius, at a release of 250 lbs per min.
VULPopulation=Residential
VULPublicReceptors=Calallen High School, Residences, Hospitals, Recreation areas, Major commercial, office.
VULCriticalFacilities=Northwest Community Hospital
VULEnvironmental=Officially designated wildlife sanctuaries
MediumVULZoneAss=N/A
MediumVULReceptors=N/A
MediumVULCriticalFacility=N/A
MediumVULEnvironmental=N/A
MediumVULPopulation=N/A
WorstVULZoneAss=Greater than 25 mile radius, with a release of 180,000 lbs. In 10 min
WorstVULPopulation=Residential
WorstVULReceptors=Calallen H.S.,Residences, Hospitals, Recreation area, major commercial, office, or industries.
WorstVULCriticalFacility=Northwest Hospital
WorstVULEnvironmental=Officially designated wildlife sanctuaries.

f. Emergency Plan
Attributes:
ResponseActions FacilityResponder
EmergencyCoordinator EvacuationPlans
DescriptionofEquipment IdentityOrgs
Disposal Procedures

Example:

```
dn : ou=emergency, cn= O.N.Stevens Water Filtration Plant, o=e-Plan, c=us
ResponseActions=Chemical Spill call into Hazmat team
EmergencyCoordinator=Operations supervisor
DescriptionofEquipment=SCBA, level A, and level B suits
DisposalProcedures=Plant manager, and contract
FacilityResponder=N/A
EvacuationPlans=Through front gate
IdentityOrgs=Ed Garana Water Superintendent (361)857-1887
```

g. Employee Training

**Attributes:**
- EvacuationRoutes
- Procedures
- CriticalProcedures
- TraininginProcess
- MSDS
- EmerNotificaiton
- EmerIdentification
- AlertingProcedures
- FacilityResponder
- RescueMedicalDuties

Example:

```
dn : ou=employee, cn= O.N.Stevens Water Filtration Plant, o=e-Plan, c=us
EvacuationRoutes=Through Front gate
Procedures=Each shift supervisor is responsible for accounting for personnel
CriticalProcedures=Operator in pressured control room
TraininginProcess=City of Corpus Christi, Water Department, Sec. 2
MSDS=On site in lab.
```

h. Site Cleanup Contractor

**Attributes:**
- BusinessName
- ContractNumber
- PhoneNumber

Example:

```
dn : ou=contractor, cn= O.N.Stevens Water Filtration Plant, o=e-Plan, c=us
BusinessName=Miller Environmental Consulting Services
PhoneNumber=3612899800
ContractNumber=N/A
```
4.5 Interface Scripts Development

The ICP database interface uses a set of Common Gateway Interface (CGI) scripts and HTML files to interact between the server and clients. Users are able to connect to the server and make queries through browsers like Netscape or Internet Explorer. These CGI scripts are written using the Perl CGI module, which includes many convenience functions for writing HTML forms. The other Perl module used in the project is Net::LDAP, which provides an object oriented interface to the LDAP servers. There are several advantages of using the Net::LDAP module.

- By using the Perl object interface, the perl-ldap modules provide programmers with an interface which allows complex searches of LDAP directories with only a small amount of code.
- All the perl-ldap modules are written entirely in Perl, which means that the library is truly cross-platform compatible.

The following section presents functional descriptions of the modules, which will be used to implement the web-based ICP database.

Authorization Script: a script to check for an authenticated user.

Name: login.cgi

Input: login and password from the text field.

Function: in this module the user is prompted to enter a distinguished name (DN) and a password to view or modify the protected information. If the user is authorized, he/she is led to the related web page. The module creates a cookie of a random 10 digit number in the user’s local machine as session number. Thus every time that
the user tries to run any CGI script located in the server, the script will check if
the user machine has the session number. The script will exit before executing any
operation to the database if it fails to get the session number.

Output: if either login or password field is empty, the login.cgi will generate a form
which consists of two text fields and one submit button. After a user enter the
login and password, the script will generate an HTML page that contains the main
menu for the user. The page contains the following information: search database,
add a new entry, modify an existing entry, delete an entry, generate a report, and
online help.

Search Script: a script for searching an entry in the database.

Name: searchentry.pl

Input: facility name as distinguished name and sub section name like chemical
inventory, employee training, etc.

Function: in this script the submit button invokes the searchentry.pl running in the server
side to fetch the related information according to the facility name and sub section
name.

Output: an HTML page containing every entry name and a list of the attributes in the
entry.

Add script: a script for adding a new entry.

Name: addentry.pl

Input: facility name and other information about the facility from addentry html pages.
Function: in this script, the user is requested to first enter the facility name and then fill in a form with many blank input fields about general facility information. Consequently, the user is able to enter other facility information about chemical inventory, employee training, vulnerability analysis, emergency plan, site cleanup contractor, etc. After the user submits the facility name and, the script will set another cookie of FacilityName (FacilityName is common name for a facility and required in the individual entry of directory server.) and meanwhile it saves the information in the database and generates a new record.

Output: confirmation message about the new record.

Modify script: a script for modifying an existing module.

Name: modifyentry.pl

Input: facility name and any attribute that the user would like to modify from modifyentry html pages.

Function: first, the user is requested to select a facility name, which he/she would like to modify. Then the user is led to the main menu where he/she is able to choose a subsection. Every subsection in the modify module has the same graphic user interface as the add module. The script returns the current information in the database to the user side. The user is able to modify the entry in the input field and click the submit button, which runs the modify script to replace the existing attributes.

Output: confirmation of the modification.
Generate Report Script: a script for generating reports for an existing facility.

Name: generate-report.pl

Input: the facility name and report name.

Function: the user needs to select the facility name and report name. Then it runs the script to generate one report every time.

Output: the formatted report of the facility that the user chooses.

4.6 Testing the e-Plan System

Now that the e-Plan system is implemented, testing of all modules and attributes can be performed. Different users with different privilege will enter the e-Plan system to test the security. Records will be added, updated and deleted for each field in each form from the browser. Every Perl script will be tested through both a Linux command line and the browser. All types of reports will be to verify the regulatory requirements.

After the e-Plan system is tested and debugged and the EPA region VI is satisfied with it, the final product will be installed at National Oil Spill School and serve the whole Corpus Christi Area.

4.7 Online Help/User Interface Design

The focus of this project has been the development of LDAP entity attributes and program interfaces for implementation of the e-Plan system. Some components of the system such as user interface design and an online help system have been left as future improvements of the system.
5. RESULTS

This e-Plan has been implemented by using server/client programming and database management technologies. The e-Plan is used as the planning tool to add, modify, delete, search and generate reports through the Internet or Intranet. The e-Plan has consistent and simple interfaces to allow the clients to do different queries and grant the privileges to other users. However, in order to optimize the e-Plan in the real-world, some features need to be added or improved in the future. First of all, clients will be able to write their own query to search the whole database. For example, the fire department can find the detail that they need to know during the emergency. Second, multiple servers will be installed in different location. Currently, the central server stores every facility’s information and only the e-Plan administrator will have full control of the system. Later, every facility will be able to control its own information by installing a local server.
6. REFERENCES

1. Wilcox, Mark Implementing LDAP.


3. Tom Christiansen, Nathan Torkington. Perl Cookbook

4. Ben Laurie, Peter Laurie. Apache The Definition Guide

5. Isabelle N. Michaud, Graduate Project- The Design and Implementation of the
   Natural Sounds Collection Database System. Texas A&M University ---
   Corpus Christi, Texas, 1997.

   Essential Technologies, Inc. Rockville, MD. 1999

7 Perl official site. Available from www.perl.com
   (cited Dec. 10, 1999)

8 OpenLDAP official site. Available from www.openLDAP.org
   (cited Dec. 10, 1999)

   (cited Dec. 10, 1999)
#!/usr/bin/perl

use CGI;
use CGI::Valid qw(:standard);
use CGI::Carp qw(fatal_level);$\_;
use URI::URL;
use Net::LDAP;

my $query = new CGI;
my $www_host = "localhost";
my $www_port = 80;
# The parameters for the search
my $ldap_host = "localhost";
my $ldap_port = 389;
my $ldap_base = "ou=ICP Project,C=us";
my $dn = "";
# Check if the client has the cookie.
@http_pairs = split(/; /, $ENV{"HTTP_COOKIE"});
foreach @http_pairs {
    (my $name, my $value) = split(/=/, $par);
    %cookie($name) = $value;
}
if (!cookie("sessionID")) {
    print "Content-type:text/html
"
    print "<HTML>
    <head><title>Access Denied</title></head>
    <body>
    <center>
    <font size=5>You are not allowed to search!</font></center>
    </body>
    </HTML>
exit 0;
}

if ($query->request_method() eq "GET" || $query->param("name1")) {
    print $query->header;
    print $query->start_html("title" => "Search Form");
    print $query->xhtml("Search by Facility Name");
    my $action = $query->url;
    print $query->startform("method" => "POST",
                            "action" => $action,
                            );
    print "<form border=0>
    print "<TR><TD>Please enter your facility name</TD><TD>:</TD>
    print $query->textfield("name1" => "name1",
                            "default" => "",
                            "size" => 50,
                            );
    print $query->br();
    print "<form>Choose the sub section you would like to make a search:
    print $query->br();
    print "<UL>
    print "<LI>"all", "entire facility", "chemical", "chemical inventory", "vulnerability", "vulnerability analysis", "employee", "employee training", "emergency", "emergency plan", "cleanup", "site cleanup contract or agreement"
    print $query->hr();
    print $query->scrolling_list("name" => "section",
                               "value" => ["all", "chemical", "vulnerability", "employee", "emergency", "cleanup"];
    print $query->.hr();
    print $query->submit();
    print $query->endform();
    print $query->end_html();
}

else {
    print $query->header;
    print $query->start_html("title" => "Facility General Information");
    my $fname = $query->param("fname");
    my @section = $query->param("section");
    my $ldap_host = "localhost";
    my $ldap_port = 389;
    my $ldap_base = "ou=ICP Project,C=us";
    my $dn = "";
    my $pwd = "";
    my $ldap = new Net::LDAP($ldap_host);
    $ldap->bind( # bind a directory by dn and password.
               dn => "cn=root, ou=ICP Project, c=us",
               password => "secret"
           );
    if (@section[0] eq "all") {
        $msg = $ldap->search(
            base => $ldap_base,
            filter => "objectClass",
        ) || die ("Failed on search. $!");
    } else {
        foreach @section {
            $ldap_base = "ou=$section, $ldap_base;
            $msg = $ldap->search(
                base => $ldap_base,
                filter => "objectClass",
            ) || die ("Failed on search. $!");
        }
    }
    $ldap->unbind;
    print $query->xml("Your Facility Information:");
    if ($msg->all_entries) {
        print "$fname did not return any results"
    } else {
        foreach $entry ($msg->all_entries) {
            print $entry->dn;
            foreach $attr ($entry->attributes) {
                print "$attr = "$;
            }
        }
    }
    print $query->end_html();
}
#!/usr/bin/perl
use Net::LDAP;
use CGI;
use CGI qw(CGI :standard);
use CGI::Carp qw(fatalError);

# check if the client has the cookie
$selfpairs = split(/; /, $ENV{'HTTP_COOKIE'});
foreach $pair (@$selfpairs) {
    ($name, $value) = split(=//, $pair);
    $cookie($name) = $value;
}

if (!defined $cookie('sessionID')) {
    print <<EOR;
    <html>
    <head> <title> Access Denied </title> </head>
    <body>
    <center>
    <font size=5> You are NOT allowed to add an entry!</font>
    </center>
    </body>
    EOR
    exit 0;
}

my $objectclass = 'building';
my $buildingName = param('BuildingName');
my $HVACEquipment = param('HVACEquipment');
my $scrubberExhaust = param('scrubberExhaust');
my $AccessHatches = param('AccessHatches');
my $SmokeHatches = param('SmokeHatches');
my $Skylights = param('Skylights');
my $crosssectional = param('crosssectional');
my $ConstructionType = param('ConstructionType');
my $localSelStorage = param('localSelStorage');
my $indification = param('indification');
my $quantitiesInStorage = param('quantitiesInStorage');

$ldap = new Net::LDAP('pc212.cbi.tamu.edu');
$cn = $cookie('FacilityName');
$dn = 'ou=building,cs='.$cn.','.$IP . 'Project, cn=US';

$ldap->bind {
    dn => 'cn=root, o=IP Project, c=us',
    password => 'secret'
};

$msg = $ldap->add {
    dn => $dn,
    attr => {
        'objectclass' => $objectclass,
        'buildingName' => $buildingName,
        'HVACEquipment' => $HVACEquipment,
        'scrubberExhaust' => $scrubberExhaust,
        'AccessHatches' => $AccessHatches,
        'SmokeHatches' => $SmokeHatches,
        'Skylights' => $Skylights,
        'crosssectional' => $crosssectional,
        'ConstructionType' => $ConstructionType,
        'localSelStorage' => $localSelStorage,
        'indification' => $indification,
        'quantitiesInStorage' => $quantitiesInStorage
    }
} || warn 'failed to add entry. $!';

$code = $msg->code;
$ldap->unbind;

$forward = 'http://pc212.cbi.tamu.edu/project/addentry/chemical/chemical.html';
print <<EOR;
Content-type: text/html

<html>
<head> <title> Selection after submit </title> </head>
<body>
You just entered the building information! <br/>
</body>
</html>
EOR
exit 0;
#!/usr/bin/perl

use Net::LDAP;
use CGI;   
use CGI:: qw(CGI :standard);
use CGI:: Carp qw(fatalToBrowser);

# check if the client has the cookie
$nvpairs = split(/\; /, $ENV{'HTTP_COOKIE'});
foreach $pair (@$nvpairs) {
  ($name, $value) = split(/=\s*/, $pair);
  $cookie($name) = $value;
}

if (!$cookie('sessionID')) {
  print "<html><head><title>Access Denied</title></head>
<body>
<center>
<html size=5>Member is NOT logged in allowed to add an entry!</html>
</center>
</body></html>

EOR  
exit 0;
}

my $objectclass = 'chemical';
my $ ChemicalName = param('ChemicalName');

$ldap = new Net::LDAP("pc212.cbi.tamucc.edu");
$cn = $cookie('FacilityName');

$ldap->bind {
  dn => "cn=root, ou=ICP Project, c=us",
  password => 'secret';
};

$dn = "ou=chemical, cn=", $cn, "c=ICP Project, c=us";
$msig = $ldap->add {
  dn => $dn,
  attr => [
    'objectclass' => $objectclass,
  ] } || warn "failed to add entry. $!";

$dn = "cn=", $ChemicalName, ", cn=", $dn;
$msig = $ldap->add {
  dn => $dn,
  attr => [
    'objectclass' => $objectclass,
    'chemicalName' => $ChemicalName
  ] } || warn "failed to add entry. $!";

$code = $msig->code;
$ldap->unbind;

# $forward = "http://pc212.cbi.tamucc.edu/project/addentry/chemical/chemical.html";
print "<html><head>
Content-Type: text/html

<body>
You just entered a chemical, please choose the form in the left column you would like to continue<br>
</body>
</html>

EOR  
exit 0;
#!/usr/bin/perl

use Net::LDAP;
use CGI;
use CGI::qW(CGI => 'standard');
use CGI::Carp qw(fatalErrors);

# check if the client has the cookie
$nvpairs = split(/; /, $ENV{'HTTP_COOKIE'});
foreach $pair (split(/=/, $nvpairs)) {
    ($name, $value) = split(/=/, $pair);
    $cookie($name) = $value;
}

if(!$cookie('sessionID')) {
    print "Content-Type:text/html

     <html><head><title>Access Denied</title></head>
     <body>
     <center>
     <font size=5>You are NOT allowed to add an entry!
     </center>
     </body></html>

    exit 0;
}

my $objectclass = 'emergency';
my $employeeresponsesactions = param('EmployeeResponseActions');
my $emergencycoordinator = param('EmergencyCoordinator');
my $discriptionofemergencyequipment = param('DiscriptionOfEmergencyEquipment');
my $disposalprocedure = param('DisposalProcedure');
my $onsitefacilitycoordinator = param('OnsiteFacilityCoordinator');
my $evaluationplans = param('EvaluationPlans');
my $identifyindividual = param('IdentifyIndividual');

$ldap = new Net::LDAP('pc212.cbi.tamu.edu');
$cn = $cookie('FacilityName');
$dn = ou="emergency", cn="$cn", o=ICP Project, c=us;

$ldap->bind(
    dn => 'cn=root, o=ICP Project, c=us',
    password => 'secret');

$msg = $ldap->add(
    dn => $dn,
    attr => [
        'objectclass' => $objectclass,
        'employeeresponsesactions' => $employeeresponsesactions,
        'emergencycoordinator' => $emergencycoordinator,
        'discriptionofemergencyequipment' => $discriptionofemergencyequipment,
        'disposalprocedure' => $disposalprocedure,
        'onsitefacilitycoordinator' => $onsitefacilitycoordinator,
        'evaluationplans' => $evaluationplans,
        'identifyindividual' => $identifyindividual,
    ]); # warn 'failed to add entry. $?';

$code = $msg->rcode;
$ldap->unbind;

#$forward="http://pc212.cbi.tamu.edu/project/addentry/chemical/chemical.html";
print <<=EOJ;
Content-type: text/html

<html><head><title>Selection after submit</title></head>
<body>
You just entered an emergency plan, please choose the form in the left column you would like to continue<br>
</body>
<html>
EOJ
exit 0;
#!/usr/bin/perl

print 'Content-type: text/html

if($nvpairese) {
  print qq{
    <head>  <title>Main Menu</title>  </head>
    <body bgColor="#FF0000">
      <h1 align="center"><font color="#00FF00">Modify an Existing Entry</font></h1>
      <form action="/cgi-bin/modify.pl" method="post"

  else {
    print qq{
      <html>
        <head>  <title>Access Denied</title>  </head>
        <body>
          <center>
            <font size=5>You are not allowed to access this page!</font>
          </center>
        </body>
      </html>
    exit 0;
  }

my $query = new CGI;
my $query->cookie('sessionID')

if($query->request_method() eq 'GET' || $query->param('fname')) {
  print $query->header;
  print $query->start_html(-title="Search Form");
  print $query->hl("Enter Your Facility Name Before Modification.");
  my $action = $query->url;
  print $query->startform(-method='POST',
    -action=>$action,
  );
  print "Please select your facility name:"

  print $query->br();
  my $ldap_host = "$localhost"
  my $ldap = new Net::LDAP("$ldap_host");
  $msg = $ldap->search(
    base => "o=ICP Project,c=us",
    scope => 'one',
    filter => 'objectclass=facility'
  ) || die("Failed on search. $!");
  foreach $entry (@$msg->all_entries) {
    @entry[1], $entry[0] = split(/ /, $entry[0]);
    if ($entry[1] eq "") { $fname = $1;
    if ($fname ne "") { push @$entry[1], $fname, $fname[1];
  }
  print $query->br();
  print $query->scrolling_list(-name => 'fname',

  print $query->p();
  print $query->submit("Next");
  print $query->endform();
  print $query->end_html();

  else {
    my $facilityName = param("fname");
    if ($facilityName) {
      $forward = "http://pc212.cbi.tamucc.edu/project/modifyentry/facility/facility_page.html"
      print "Content-type: text/html"
      print "Set-Cookie: FacilityName=$facilityName; path=*
";
print "Wn"
print "<EOR>";
<html><head><META HTTP-EQUIV="REFRESH" CONTENT="0;URL=$forward;">
EOR
</head></html>

You have chosen the facility name!</a>

exit 0;
}

my $query = new CGI;

# Check if the client has the cookie.
my $nvpairs = split(/;/, $ENV{'HTTP_COOKIE'});
foreach $pair (@nvpairs) {
  ($name, $value) = split(/=/, $pair);
  $cookie{$name} = $value;
}

if (!$cookie{'sessionID'}) {
  print "Content-type: text/html
"
  print "<EOR>";
  $cookie = undef;
  print "<html><head><title>Access Denied</title></head>
<body>
<center>
You are not allowed to search!
</center>
</body></html>
EOR
exit 0;
}

if ($query->{_request_method} eq "GET" || ! defined($query->{_param{'name'}})) {
  $query->{_header} = undef;
  $query->{_start_html} = "Search Form";
  $query->{_hi} = "Enter Your Facility Name Before Modification:";
  my $action = $query->{_url};
  $query->{_startform} = "method='POST',
    -action=>$action.
  ";
  print "<table border=0>
"
  print "<tr><td>Please enter your facility name: </td>
"
  print "<td><input name='name',
    -default=''
    -size=50>, </td>
"
  print "<tr><td><table>
"
  print "$query->{_br}();
  print "$query->{_p}();
  print "$query->{_submit}('Next');
  print "$query->{_endform}();
  print "$query->{_end_html}();
  }

else {
  my $facilityName = $query->{_param{'name'}};
  my $ldap_host = "localhost"
  my $ldap = new Net::LDAP("$ldap_host");
  $msg = $ldap->search(
    base => 'o=ICP Project, c=us',
    scope => 'one',
    filter => "(&(objectClass=FacilityName=$FacilityName)"
  ) || die ("Failed on search. $!");
  if ($msg->{_entries}) {
    $forward = "http://www.cbi.tumc.edu/project/modifyentry/facility/facility.html";
    print "Content-type: text/html"
    print "<Set-Cookie: FacilityName=$FacilityName; path=/>
    print "Wm"
    print "<EOR>"
    $html = "<META HTTP-EQUIV="REFRESH" CONTENT="0;URL=$forward;">
EOR
"a href='"forward'"">You have entered the facility name!</a>
#!/usr/bin/perl

use Net::LDAP;
use CGI;
use CGI qw(CGI :standard);
use CGI qw(fatalToBrowser);

# check if the client has the cookie
$nvpairs = split(/; /, $ENV{'HTTP_COOKIE'});
foreach $pair (@nvpairs) {
    @nvpairs = split(/=/, $pair);
    $cookie($nvpairs{0}) = $nvpairs{1};
}

if (!defined($cookie{'sessionID'})) {
    print "Content-type: text/html\n\n";
    print <<EOR;
    <html><head><title>Access Denied</title></head>
    <body>
    <center><font size=5>You are NOT allowed to add an entry!</font></center>
    </body>
    </html>
EOR
    exit 0;
}

my $FacilityName = $cookie{'FacilityName'};
if (defined(param('ChemicalName'))) {
    $items{'chemicalName'} = param('ChemicalName');
}

$ldap = new Net::LDAP('pc212.cbi.tamu.edu:389');
$cn = $cookie{'FacilityName'};

$ldap->bind {
    dn => 'cn=root, o=ICP Project, c=us',
    password => 'secret';
};
$dn = "cn=".$chemicalName.". ou=chemical, cne= $cn,".$chemicalName.". o=ICP Project, c=us";

$ms = $ldap->modify (dn => $dn, replace => %items)
    || warn "failed to add entry. $!";
$code = $ms->code;
$ldap->unbind;

# forward="http://pc212.cbi.tamu.edu/project/addentry/chemical/chemical.html"
print "Content-type: text/html\n\n";

<html><head><title>Selection after submit</title></head>
<body>
You just modified a chemical, please choose the form in the left column you would like to continue<br>
</body>
</html>
EOR
exit 0;
#!/usr/bin/perl

use CGI;
use CGI qw(:standard);
use CGI::Carp qw(fatalError);

#check if the client has the cookie
@nvpairs=split(/; /,$ENV{'HTTP_COOKIE'});
foreach @nvpairs { 
  ($name,$value) = split(/=/,$pair);
  $cookie{$name} = $value;
}

if(!(defined $cookie{'sessionID'}))
{
  print "Content-type: text/html\n\n";
  print <<EOR;
  <html><head><title>Access Denied</title></head>
  <body>
  <center>
  <font size=5>You are NOT allowed to add an entry!</font>
  </center>
  </body></html>
EOR
exit 0;
}

my $FacilityName = $cookie{'FacilityName'};
if(param('CleanupName'))
{
  $items = param('CleanupName');
}
if(param('CleanupPhoneNumber'))
{
  $items = param('CleanupPhoneNumber');
}

$ldap = new Net::LDAP("pc212.cbi.tamu.edu");
$nCn = $cookie{'FacilityName'};
$dn = "cn=cleanup,cn=cn, cn=ICP Project, cn=us";

$ldap->bind(
  dn => $dn,
  password => "secret";
);

$msg = $ldap->modify(
  dn => $dn,
  replace => 
    { $items })
  || warn "failed to add entry $!";

$code = $msg->

$ldap->unbind;

# forward the page to the next step
print "Content-type: text/html\n\n";
print <<EOR;
  <html><head><title>Selection after submit</title></head>
  <body>
  You just entered a cleanup entry, please choose the form in the left column you would like to continue<br>
  </body>
EOR
}

else
{
  my $fname = $query->param('fname');
  $dn = "cn=, $fname,cn=ICP Project, cn=us";
  my $section = $query->param('section');
  my $facility = $query->param('facility');
  my $description = $query->param('description');
  my $time = $query->param('time');
  my $priority = $query->param('priority');
  
  print "Content-type: text/html\n\n";
  print <<EOR;
  <html>
  <form name="form">
  <input type="text" name="fname" value="$fname">
  <input type="text" name="facility" value="$facility">
  <input type="text" name="description" value="$description">
  <input type="text" name="time" value="$time">
  <input type="text" name="priority" value="$priority">
  
  <input type="submit" value="Submit">
  
  </form>
  </html>
EOR
";
}
#now do the delete
my $ldap = new Net::LDAP($ldap_host);

$ldap->bind( # bind a directory by dn and passwd.
   dn => 'cn=one,ou=ICP,ou=ICP Project,c=us',
   password => 'secret',
);

if ($section{0} eq 'all') {
   $msg = $ldap->delete($dn) || die ('Failed on delete. $!');
} else {
   foreach $section (@{section}) {
      $dn = $section{0} . $section{1}, $section{2};
      $msg = $ldap->delete($dn) || die ('Failed on delete. $!');
   }
}

$ldap->unbind;
print $query->start_html(-title=>'Delete Confirmation!');
print $query->h2('Your facility $name has been deleted!');
print $query->p('Sorry! You are not allowed to view this page!');
print $query->end_html();

if (!($query->{_request_method} eq 'GET' || $query->{_param}{name})
   print $query->header;
   print $query->start_html(-title=>'Delete Form!');
   print $query->h1('Delete Facility by Facility Name');

   my $action = $query->{_url};
   print $query->startform(-method=>'POST',
      -action=>$action,
   );
   print '<Table border=0>\n';
   print '<TR><TD>Please enter your facility name: </TD><TD>: </TD>';
   print $query->{_textfield}(-name=>'name',
      -default => '',
      -size => 50,
   );
   print '<TR><TD>Please select sub section: </TD><TD>: </TD>
   print $query->{_selected_list}(-name => 'section',
      -value => [ 'all', 'chemical', 'vulnerability', 'employee' ],
      -default => 'all',
      -multiple => 'true',
      -labels => \%list_label,
   );
   print $query->{_submit}();
   print $query->{_br}();
   print $query->{_submit}();
   print $query->{_endform}();
print $query->end_html();
}
else
{
print $query->start_html(-title=>'Delete Confirmation!');
my $fname = $query->param('fname');
my $section = $query->param('section');
my $ldap_host = $query->param('ldap_host');
my $ldap = new Net::LDAP($ldap_host);
$ldap->bind( # bind a directory by dn and passwd.
    dn => 'cn=Root, o=ICP Project, c=us',
    password => 'secret',
);
if ($section[0] eq 'all')
{
    $msg = $ldap->search(
        base => $dn,
        filter => 'objectclass=*
    ) || die ('Failed on search $msg');
    foreach $entry (@$entry)
    {
        push (@dns, $entry->dn);
    }
}
else
{
    foreach $section (@$section)
    {
        $dn = $section . $section . $dn;
        $msg = $ldap->search(
            base => $dn,
            filter => 'objectclass=*
        ) || die ('Failed on search $msg');
        @entries = ($entries, $msg->all_entries);
    }
}
foreach $entry (@entries)
{
    push (@dns, $entry->dn);
}
foreach $dn_d (reverse sort (length $a <=> length $b) @dns)
{
    $msg = $ldap->delete($dn_d);
}
$ldap->unbind;
print $query->h2('Your facility - $fname has been deleted!');
print $query->p('"$dn, $section"');
print $query->end_html();}
use CGI qw(:standard);
use CGI::Carp qw(fatalDepToBrowser);
use URI::URL;
use Net::LDAP;

my $query = new CGI;
my $www_host = "localhost";
my $www_port = 389;

# Check if the client has the cookie.
@npairs = split(/=/, $ENV{HTTP_COOKIE}, 1);  
foreach $pair (@npairs) {  
    ($name, $value) = split(/=/, $pair);  
    $cookie{$name} = $value;
}

if ($cookie{'sessionID'}) {
    print "Content-type: text/html
"
    print "<html><head><title>Access Denied</title></head>
";
    print "<center>
";
    print "<font size=-5>You are NOT allowed to access this page!</font>
";
    print "</center><body></html>
"
    exit 0;
}

if ($query->request_method() eq 'GET') {
    print $query->header;
    print $query->start_html(-title=>'Begin to generate a report');
    print $query->hi("Generate Report");
    my $action = $query->url;
    print $query->start_form(-method=>'POST',  
        -action=>$action,  
        -class=""  
    );
    print "<Table border=0>
";
    print "<TR><TD>Please enter your facility name</TD><TD>:
";
    print $query->textfield(-name=>'name',  
        -default=>'',  
        -size=>50 , "<TD><TR><TABLE>
";
    print $query->br();
    print "Choose the report name:
";
    print $query->br();
    $list_table = ('fp', 'EPA FPFR', 'nt', 'NRT One Plan Fall Report', 'mp','Risk Management Report Submit');
    print $query->scrolling_list(-name=>'report',  
        -value => ['fp', 'nt', 'mp'],
        -default => 'fp',
        -multiple => 'false',
        -labels => $list_table,
    );
    print $query->p();
    print $query->submit();
    print $query->end_form();
    print $query->end_html();
    $list_table = ('fp', 'EPA FPFR', 'nt', 'NRT One Plan Fall Report', 'mp','Risk Management Report Submit');
    print $query->scrolling_list(-name=>'report',  
        -value => ['fp', 'nt', 'mp'],
        -default => 'fp',
        -multiple => 'false',
        -labels => $list_table,
    );
    print $query->p();
    print $query->submit();
    print $query->end_form();
    print $query->end_html();
}
else {
    print $query->header;
    print $query->start_html(-title=>'Under Construction');
    print '<H1 align="center">Sorry, this report is still under construction</H1>';
}

my $fname = $query->param('fname');
my $report = $query->param('report');
#!/usr/bin/perl

use CGI;

# Address to this script.
$ThisScript = "http://pc212.chi.tamu.edu/cgi-bin/login/login.cgi";

# Page that the users should be sent to when they put in the correct
# password.
$Forward = "http://pc212.chi.tamu.edu/cgi-bin/menu.pl";

# Name of the page that users are logging into.
$Spagename = "ICP Project";

# Password required to login.
$Pass = "login";
# UNIX path to the password file on my system
$Spfile = "/home/httpd/cgi-bin/login/pwd.dat";

# Send mail to YOU when someone logs in?
$SendMail = 0;

# UNIX path to the mail program on your system.
# This program must be able to take the -o option to send mail. Such as:
# elm, Mail, etc. If you run into problems, turn mail sending off.
$Mail = "mail";

# Email address to send mail to (your personal e-mail address.)
# You MUST put a backslash (\) in front of the 'at' (@) sign in the e-mail
# address.
$ToMail = "jeff@pc212.chi.tamu.edu";

# Do you wish to log logins? (1/0)
# LOG file is NOT auto cleared. You will have to edit it by hand. If you
# delete it, remember to chmod the new file ??? when you re-make it.
$Log = 1;

# Ask for name? (Will be logged.)
$name = 1;

# Ask for an e-mail address? (Will be logged.)
$email = 0;

# What is the address to the log file? (Remember to create the file and
# to chmod it ???)
$LogFile = "/home/httpd/cgi-bin/login/log";

# Path to your system's date program for logging.
$DateProg = "/bin/date";

# Settings for login page colors.
$Text = "#cccccc";
$link = "#cccccc";
$login = "#cccccc";
$bgcolor = "#cccccc";

# generate a random ID between 0 to 10,000,000,000
$SessionID = int(rand(10000000000)) + 1;

# error("name");
if (($email == 1) && ($in('email') eq "") )
  error("E-Mail address");

unless ($in('email') =~ m/^@/i) {
  if ($email == 1) {
    error("correct e-mail address");
  }
}

if (! -e "$Log_file") {
  open(FILE, ">$Log_file")
  print FILE "File START $date\n"
  close(FILE);
}

open(FILE, ">$Log_file")
print FILE "Login: $ENV['REMOTE_ADDR'] (with $ENV['HTTP_USER_AGENT']) $date\n"
if ($name == 1) {
  print FILE " Name: $in('name')\n"
}
if ($email == 1) {
  print FILE " E-mail: $in('email')\n"
}
print FILE "SessionID $in('pass') $in('name')\n"
close(FILE);

$continue;

sub print_login {
  print "Content-type: text/html\n\n"
  print " <html> <head> <title>Login</title> </head> <body bgcolor="#ccc" text="#ccc" link="#ccc" vlink="#ccc" alink="#ccc" ctext="#ccc" > <font size=5>Please login to <b>$spagename</b>: </font> </center> </html>"

  if ($name == 1) {
    print " Name: <input type=text name=name><br><\n"
  }
  if ($email == 1) {
    print " E-mail Address: <input type=text name=email><br><\n"
  }
  print " <html> Password: <input type=password name=pass><br><\n<input type=submit value=Submit name=submit> </form> <br> Currently you are able to login as "guest". <br> Just type "guest* in the login textfield and *"password" in the password field. </p> </body> <html>"
return 1;
}

sub getcgi {
  local($In, $n);
  local($name, $value);

  # If REQUEST_METHOD is POST, use CONTENT_LENGTH. Else, use QUERY_STRING.
  if ($ENV{'REQUEST_METHOD'} eq "POST") {
    if ($ENV{'CONTENT_TYPE'} =~ m/;application/x-www-form-urlencoded/i) {
      read(STDIN, $in, $ENV{'CONTENT_LENGTH'});
    }
  }
...
{ } 

else { 
  $in = $ENV{'QUERY_STRING'};
  # Resolve and unescape name/value pairs into %in
  foreach ($split('&', $in)) {
    s/%3A/;/g;
    ($name, $value) = split('=', $_, 2);
    $name =~ s/%[0-9a-z]//ig;
    $value =~ s/%[0-9a-z]//ig;
    %in{$name} = $value;
  }
  return %in;
}

sub err {
  local($Error) = $_[0];
  print "Content-type: text/html\n\n";
  print <<<HTML;
<body><head><title>Error</title></head>
<html><center><font size="3">Error</font><p>
You must enter a $Error!
</center></p></html>
<font size="3">Continue</font>
</body></html>
exit;
return 1;
}

sub check_pw { 
  my ($pwfile, $loginname, $password, $line, $match);
  $fetch_root = 0;
  open(PWFILE, "&$pwfile") || die "Can't open password file";
  while(<PWFILE>) {
    chomp;
    $pwfile = split('/') unless ($pwfile);
    foreach $line ($pwfile) {
      if ($line/next) {
        ($loginname, $password) = split('/: ', $line);
        $root =~ s/[^0-9a-z]//ig;
        if ($password eq $in{'pass'}) && $loginname eq $in{'name'} { 
          $match = 1;
          if user pass the authenticatio, write down his/her name/pwd in
          #a temp file. The file name will be the same as sessionID.
        } 
      } 
    return $match;
  }
}

sub continue { 
  print "Content-type: text/html\n";
  print "Set-Cookie: sessionID=$sessionID; path=/";
  print "\n"
  print "<html><head><META HTTP-EQUIV="REFRESH" CONTENT="0;URL=$forward"></head><body>
  <a href="$forward">Click here to continue</a></body>";
  print "\n"
}

Tuesday April 18, 2000