

Great Plains Free-Net Inc.

Great Plains Free-Net Inc.

User Management System

By: Daryle Niedermayer

April 19, 2003

User Management System

Contents

General Project Requirements	1
<i>Project Justification.....</i>	<i>3</i>
<i>Business Case.....</i>	<i>3</i>
<i>User Requirements.....</i>	<i>3</i>
<i>System Restrictions</i>	<i>4</i>
Business Model	4
<i>Registered Users.....</i>	<i>4</i>
<i>Individual Members</i>	<i>4</i>
<i>Institutional Members.....</i>	<i>5</i>
<i>Additional Services Available to Members.....</i>	<i>5</i>
<i>Affinity Program</i>	<i>5</i>
System Architecture	6
<i>Security Concerns</i>	<i>6</i>
<i>Implementation Concerns</i>	<i>6</i>
Phased Requirements	7
<i>Phase 1: Account Creation and Billing</i>	<i>7</i>
<i>Phase 2: User Account Management</i>	<i>7</i>
<i>Phase 3: GPFN Help Desk Management.....</i>	<i>7</i>
<i>Phase 4: Affinity Pricing Model.....</i>	<i>7</i>
<i>Current Project Scope.....</i>	<i>8</i>
Implementation Timetable.....	8
<i>Task List</i>	<i>8</i>
Implementation Requirements	9
Implementation Structure.....	9
Implementation Components	10
<i>Web Interface.....</i>	<i>10</i>
<i>Shared Libraries.....</i>	<i>11</i>
User Authentication	11
Session Credentials	11
<i>System Configuration APIs</i>	<i>12</i>
Password Change.....	12
Mail Forwarding.....	12
Mail Filtering.....	12
Reviewing Disk Quota.....	12
Data Flow Analysis.....	12
<i>Data Flow Diagrams.....</i>	<i>13</i>
Top Level Data Flow Diagram.....	13

1. Process User Application Data Flow Diagram	14
2. Process User Upgrade Data Flow Diagram	15
3. Generate Invoice Data Flow Diagram.....	16
4. Process Receipt Data Flow Diagram.....	16
5. Manage User Account Data Flow Diagram.....	17
Web Navigation Model	17
Data Model.....	20
Entity Relationship Diagram	20
Data Dictionary	21
System_Default Table.....	21
User Table	22
Member Table	23
Affinity Table (Phase 4).....	23
Services Table.....	24
Email_Alias Table.....	25
Invoice Table	25
Invoice_Item Table.....	25
Receipt Table	26
Upgrade_Request Table	26
Web Pages.....	28
user/index.html	28
user/join.html	29
user/agree.html.....	29
user/apply.html	30
user/upgrade.html	31
user/myacct.html	32
vol/index.html.....	32
vol/invoices.html	33
vol/invoice.tmpl	34
vol/stmt.tmpl.....	34
vol/rcpt.tmpl.....	35
vol/accept.tmpl.....	35
vol/acct.html.....	36
CGI Procedures	37
user/validate.cgi.....	37
user/login.cgi.....	39
user/upgrade.cgi.....	40
vol/listapps.cgi	42
vol/listupgrades.cgi	44
vol/geninvoice.cgi	46
vol/receipts.cgi	49
Shared Library Procedures.....	50
lib/uname_test.pl	50
lib/uname_generate.pl	51
lib/pw_check	52
lib/Login.pm	54
lib/auth_user	56
System Configuration Procedures	59
sys/add_account	59

sys/add_alias.....	59
sys/change_user.....	61
sys/add_virtualdomain.....	61
sys/add_ppp.....	62
sys/add_db.....	62
sys/add_quota.....	63
sys/add_ssl.....	63
sys/add_listserver.....	64

User Management System

Project Requirements

General Project Requirements

The Great Plains Free-Net Inc., based in Regina Saskatchewan, is a non-profit, volunteer-driven corporation dedicated to providing low and no-cost internet connectivity to residents, community groups and organizations that are not sufficiently served by commercial Internet Service Providers (ISPs).

In particular, the Great Plains Free-Net caters to individuals:

- whose personal computers are too old or antiquated to run current generation web-browsers and other internet client software;
- who want a text only user interface to a graphical user interface because of personal preference or disability;
- who are new to the Internet environment and who need more “hand-holding” and support that commercial ISPs are prepared to offer; or
- who are not heavy users of the Internet and who do not require the unlimited or generous usage offers characteristic of many commercial offerings.

For community groups, the Great Plains Free-Net offers:

- no and low cost web hosting to community groups seeking to inform the public of their services and programs;
- e-mail list services to cyber-communities for whom face-to-face meetings are unrealistic;
- Database hosting services for dynamic web content; and
- Virtual domain hosting on a cost-recovery basis for community groups, individuals and organizations that wish to pay for this service.

Because of the mix of products and services offered to the public, the Great Plains Free-Net (GPFN) needs a User Management System (UMS) to manage its user base. As GPFN is volunteer driven, this system needs to be managed remotely by volunteers. As GPFN exclusively uses a Linux server base, any new system should be implemented in a Linux environment and be Internet and Web enabled.

In general, the UMS needs to provide the following functions:

1. Allow users and prospective users to use the Web to apply for a new user account;
2. Allow existing users to apply to upgrade their account to include new services or increased quotas;
3. Allow GPFN volunteers to approve or reject applications for new accounts or account upgrades and, if approved, effect the necessary changes to the system;
4. Allow users some access to their own system configuration such as changing their password, reviewing their account history, changing mail forwarding settings, and possibly adjusting spam filters;
5. Allow GPFN volunteers access to change the configuration parameters listed in function (4) above for any GPFN user;
6. The GPFN charges for memberships and other services based on a calendar year. The system must support a "renewal cycle" whereby all existing GPFN users are asked to explicitly renew their account. This renewal is validated by responding to an e-mail notification (in the case of free accounts), or by remitting a cheque for any memberships or service fees. This renewal cycle is generated through a scheduled task (or "cron job") or via a GPFN volunteer's explicit action;
7. The renewal cycle must not generate renewals for accounts which are not in arrears and must be able to generate a paper invoice or statement upon request;
8. The system must allow GPFN volunteers to process receipts against outstanding renewal requests so that user accounts are not again flagged for renewal notification during the current renewal or billing cycle;
9. The system must allow GPFN volunteers to expire or suspend accounts for any reason such as a user misusing their account or not responding to a renewal request. If a user remedies the reason for the suspension, the system must allow a GPFN volunteer to reinstate a previously suspended account. Alternatively, the system must also allow GPFN volunteers to delete any account previously expired;
10. In order to support expirations due to non-renewal, the system must be able to offer a "batch mode" whereby all non-renewed accounts are listed and expired together. Similarly, once a decision is made to delete accounts, the option should exist to allow a volunteer to delete all expired accounts as a batch process;
11. The system must maintain a history of the financial transactions against an account including the date and method by which a renewal notification was sent (either e-mail or printing), and the date and method by which a payment was received (cheque, cash, donation in kind, or credit card);
12. For audit purposes, the system must track the GPFN volunteer who performed any action.

Project Justification

To date, GPFN has used a software package called Chebucto Suite to perform all User Management functions. The Chebucto Community Network in Halifax, Nova Scotia wrote Chebucto Suite (or Csuite). It uses a collection of about 300 shell scripts and a series of flat file datastores to manage all aspects of a user's account on the system. Because of the changing nature of the Internet market space, Csuite development is now halted as is system support and maintenance.

Csuite assumes that all users will interact with the system using the Csuite shell or "Cshell". This Cshell is a restricted, custom compiled version of a lynx web browser. Lynx is a text-only web browser that renders HTML pages without displaying any graphics. The Csuite version of Lynx allows users to use keystrokes to branch off into a file editor, a mail reader (pine), as well as manage or review aspects of their user.

Csuite was written for RedHat Linux 4.1. GPFN is planning to upgrade to RedHat Linux 7.2 for its new server. A number of security concerns, new features, and system libraries predicate the need for this upgrade. However, in initial examination of the Csuite package, it is likely that any attempt to migrate Csuite to this new Linux platform would necessitate a line-by-line review of the existing Csuite code. Such a review would be very costly in terms of volunteer personnel resources and would be very specialized work. As well, GPFN did not use some Csuite features while Csuite could not support other desired features.

At the same time, new technology in existence since Csuite was initially developed means that the backend datastores can use a complete Relational Database System (RDBS) such as MySQL. The application logic can use newer more capable implementation languages. This combination of a RDBS and new implementation languages will result in a robust system that is more flexible to the business needs of GPFN and will require less development effort than that required for the rewrite of Csuite.

Business Case

GPFN is embarking on a comprehensive modernization strategy. As part of this strategy, GPFN is offering a number of new services and adopting a more member-centric philosophy. One problem with the existing system is that very few volunteers understand or can manage Csuite. Therefore member inquiries, problems or complaints often took a number of days to resolve. This delay resulted in member and volunteer frustration. At the same time, the Csuite package was hard-coded in terms of services and member categories, resulting in a collection of ad-hoc systems, databases and other methods to manage services and membership classes specific to GPFN.

While GPFN wants to maintain the Cshell for use by members and users who have older equipment, a growing number of members do not use Cshell. These members often use GPFN's PPP connection for graphical web browsing, some POP client to connect to GPFN's mail servers, or a third party high-speed internet connection to view GPFN's web resources. For these users, a web-enabled system to allow them to review their account settings and change account parameters (such as passwords) is important.

User Requirements

It is expected that GPFN volunteers would interact with the new system using a graphical web browser. However, there will be users and volunteers who prefer the Lynx browser as their user interface. For this reason, any user interface should be implemented so that it

does not make use of a graphics intensive front end. Similarly, the use of imagemaps, Javascript, applets, Shockwave and Flash media is not appropriate.

System Restrictions

Because GPFN is a volunteer-driven, non-profit organization, cost containment is always an organizational issue. The UMS must be implemented in as cost-efficient a manner as possible. This necessitates the use of open-source or other free software and development tools; conversely, GPFN discourages the use of products which require license fees or other costs.

Because GPFN's environment is exclusively based on Linux, any system must be implemented on the latest release of RedHat Linux. At the same time, because Linux is a constantly evolving operating system, any system should be sufficiently flexible and robust that it can be ported to newer releases of RedHat as those releases become available. This requirement can be met by attempting to confine operating system specific calls to a single layer within the application environment.

Business Model

During a Joint Application Development (JAD) session with GPFN board members, volunteers and members, the business model employed by the Free-Net was enunciated and elaborated.

Registered Users

The Free-Net has three basic classes of users. Registered Users are users who have signed up for service but who are not members of the corporation. Registered users are essentially "Free" users. They are not billed for their usage of the system. In return they are entitled to internet access using a GPFN dial-up account but are only offered a Cshell with its component text-only web browser and text based e-mail reader. They have 5 Mb of disk quota in the mail system and 2 Mb of disk quota for personal files and downloads in their home account. They are guaranteed one hour of usage per day.

Individual Members

Individual members have paid a membership fee (currently \$24 per annum) to become a member of corporation. The constitution of GPFN allows for the contribution of service or goods-in-kind in lieu of the membership fee. Individual members are entitled to full participation in the governance of the corporation including voice and vote at all public meetings and the opportunity to be elected to the board of directors.

From a system usage perspective, institutional members are entitled to the same access privileges as Registered Users except that they:

1. Are guaranteed unlimited dial-up access but only in one-hour blocks of time;
2. Are permitted personal web pages within their home directory;
3. Can purchase additional services for their account such as increased quota, graphical PPP dial-up access, database access, or virtual web hosting.

Institutional Members

Institutional memberships are intended for community groups and organizations comprised of more than one person. As such, the same rights, responsibilities, privileges and opportunities exist to them as for Individual members with the following exceptions:

1. They are permitted 5 Mb disk quota within the public web space of GPFN;
2. Although they are only entitled to 1 e-mail account, they can have up to four e-mail aliases to forward mail to off-site accounts;
3. The membership fee is currently \$48 per annum.

Additional Services Available to Members

A number of additional services are available to members only. These services include:

1. Increased quota for personal directory space, public web space, or incoming e-mail space;
2. PPP dial-up access so that members can dial in to the Free-Net's modem pool and use their Netscape, Internet Explorer, Microsoft Outlook or other graphical browser or POP mail client to access the Free-Net's resources;
3. Subscribing to a database within GPFN's hosted database servers;
4. Subscribing to a virtual domain web hosting service whereby the member can have a web site with the domain name of their own choosing but hosted on GPFN's servers.

The costs for some of these services are not yet determined but they are all billed on the basis of a calendar year.

Affinity Program

As a holdover from the default implementation of Csuite, a number of current GPFN memberships are held as "family" memberships. This is a membership category employed by the Chebucto Community Network but never formally adopted by GPFN. However, a small number of members found an orphaned page on GPFN's web site and sent in a cheque for \$40 for a family membership. GPFN decided to honour this membership option.

During the JAD session, GPFN stakeholders decided that they wanted any new system to have the option of supporting family memberships. Upon further discussion, it was agreed that the new system should have the capability of supporting a number of "aggregator" type accounts. Another example of such an account would be a company or non-profit group who has given a donation to the Free-Net. In exchange for the donation, GPFN would provide memberships at a reduced cost or at no cost to members of that company or group.

Such an "affinity" program would give GPFN a way of forming and cultivating strategic partnerships with other community groups and would necessitate a special pricing option for some members who also belong to the partnering organization.

System Architecture

Security Concerns

Since the implementation will use a Web Server connected to the Internet, security is a concern. This concern stems from two sources: how to prevent a “man in the middle attack” whereby a third party will intercept transmissions to and from the application and learn about a user’s password or other personal information, and how to ensure that an individual interacting with the system is entitled to perform such interactions. The former concern involves an issue of privacy, the latter an issue of authenticity.

The “man-in-the-middle” attack is not a present concern. Since GPFN has no plans to conduct on-line financial transactions at this time, there is little reason for a malefactor to attempt to hijack data transmissions from the UMS. At the same time, most users and volunteers will be using GPFN’s own modems to connect to the system further minimizing the chance for intercepting data transmissions. However, the requirement to support SSL encrypted connections may be added in the future.

The issue of user authentication is more pressing. The database should be secured so that only applications on the local host can access the data with write privileges. The application level should be designed so that user authentication is required before accessing the system and once authenticated a user’s credentials are maintained for the duration of the session. Once a session is finished or a time limit expired, these credentials must be revoked.

User authentication is important for both user access and volunteer access to the system. In the former case, it must be absolutely guaranteed that a user will never be able to access another user’s data. Similarly, it must be absolutely guaranteed that only approved GPFN volunteers will have access to the information of any user.

Implementation Concerns

The proposed system will be implemented by volunteers. This reality has a number of constituent consequences:

- Volunteers work at their own pace and may not be prepared to meet arbitrarily imposed deadlines.
- Volunteers will be working on their own time and from their own homes making it difficult to formalize project meetings or other team get-togethers.
- Volunteers have a wide range of skills and a wide range of competencies within each skill set. The design must include a variety of tasks for volunteers of every skill set and ability.
- To implement the system in the quickest manner, a phased approach should be adopted so that basic functionalities can be built and the system put into service at the earliest available date.

Implementation Phases

Phased Requirements

The development will be broken into four phases.

Phase 1: Account Creation and Billing

Phase 1 will include the minimum components required to add and support users on the new system. As such, it will include the use cases of allowing users to request new accounts, allowing users to request upgrades to existing accounts, allowing GPFN volunteers to generate a renewal cycle and process receipts against outstanding renewal requests.

Other functionality such as allowing users to check quotas or change passwords will be available through the Cshell, but not through the web enabled user interface.

Libraries of shared functions such as those required to authenticate users, set and revoke session credentials and test passwords and usernames for hardness or uniqueness are also included in this phase.

Phase 2: User Account Management

Phase 2 will allow users to manage their account and view their account history on line. They will be able to use a web interface to change their password, their mail forwarding settings, spam control settings as well as check their disk quota usage and financial history.

Phase 3: GPFN Help Desk Management

Phase 3 will allow GPFN volunteers to manage all aspects of a user's account on their behalf including setting user passwords, mail forwarding, spam control settings, and viewing or reviewing the user's disk quota and account history.

Phase 4: Affinity Pricing Model

Phase 4 will incorporate the affinity program identified in the Business Model. Account billing will then be the value of the stated membership and service charges as defined in the database, or if the account references an affinity program, the pricing used is the price referenced by the affinity entity.

Current Project Scope

Although this document will provide the general design for all four phases of the project, the detailed design specifications will be restricted to Phase 1 functionality. The additional detailed specifications for the remaining phases will be more completely scoped as they are required.

Implementation Timetable

Task List

Task	Responsibility	Phase	Required Date
Develop Visual Identity Guide, web site icons, navigation bar graphics and style sheets	Webmaster	Phase 1	Prior to web page development
Web Pages	Webmaster	Phase 1, 2 & 3	Prior to CGI Script development
Shared Library Functions	Senior Developer	Phase 1	Prior to CGI script development
Database Creation and Access Privileges	Senior Database Administrator	Phase 1	Prior to CGI script development
System Configuration Procedures	Senior Developer	Phase 1, 2 and 3	Prior to System Implementation
CGI Scripts	Senior Developer/ Webmaster	Phase 1, 2 and 3	Prior to System Implementation

System Architecture

Implementation Requirements

To satisfy the needs identified in the requirement analysis, a multi-tiered approach to the design is preferred. In particular, the components that effect the necessary changes to the system configuration based on actions taken within the UMS should be isolated from the rest of the system. In this way, a migration to a new server or a new operating system release will require changes to these components without incurring the complete system rewrite that the current migration entails.

At the same time, where multiple procedures require the same function, a list of shared libraries should encapsulate these functions for this same reason. In this way, functions and services specific to the current version of RedHat (such as user authentication) can easily be altered to take into account new features or services of future releases.

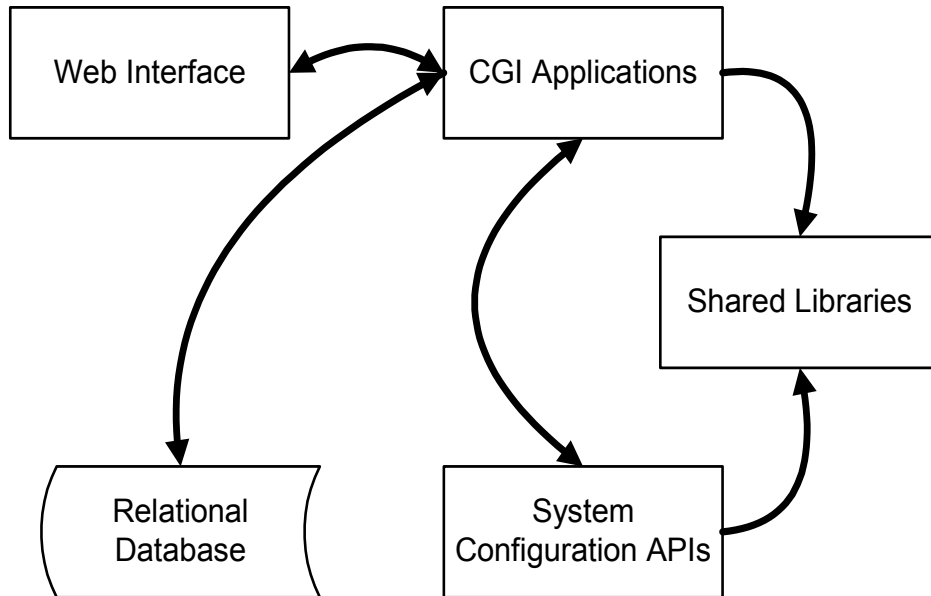
At the same, to leverage volunteer resources, the tasks of developing the web interface (a task best suited to web developers) are separated from the tasks of developing back-end functionality (a task for formal PERL programmers). This distinction has the added benefit of allowing the web interface to be updated to a new corporate look in the future without affecting the application logic.

Implementation Structure

As depicted in Figure 1 below, the develop effort can be broken into a number of conceptual parts:

- Those volunteers knowledgeable in HTML will primarily develop the Web Interface. Skills for these people should include HTML forms, graphic design and layout and site navigation expertise. A common visual theme should predominate these pieces. As well, these people will develop HTML snippets that will be incorporated by CGI Application developers when a CGI application needs to return a complete web page to the client.
 - CGI Applications will leverage people knowledgeable in PERL. These individuals should understand CGI applications, be conversant in PERL and preferably have some knowledge of SQL and how to embed database calls within PERL scripts.
 - Shared Libraries will require people very knowledgeable in PERL. Ideally, these libraries should use PERL or Object-Oriented PERL objects for functionality such as maintaining persistent user sessions.
-

- The Relational Database will use MySQL. If CGI application developers are not sufficiently conversant to embed SQL calls within their applications, an SQL-PERL specialist may perform this specialized function.
- The System Configuration APIs will likely be written in Shell script or compiled C. They will require developers with the highest skills and most likely root access to the host machine.



• Figure 1: System Conceptual Structure

Implementation Components

Web Interface

Before development of the Web Interface, a Visual Identity Guide (VIG) will be required. This guide will contain the basic style elements to ensure that all pages on the User Interface are visually and thematically consistent with each other and with the general GPFN web site.

The Visual Identity Guide will also contain directives concerning how users will navigate around the pages as well as how linkages outside the UMS pages back to the GPFN general web site will be incorporated.

In addition to the VIG, web developers will require a common set of resources such as icons, navigation bar graphics, template files and HTML Style. Developing the VIG and the other components will be the duty of the GPFN webmaster.

Shared Libraries

The Shared Library functions include all functions and methods used by two or more components of the system. It will include user authentication and managing of session credentials.

These functions will involve setting and retrieving cookies on the user's browser, writing to and retrieving session values from the relational database, and managing Object Oriented PERL constructs and entities.

User Authentication

User authentication involves comparing the username supplied on a web form with the existence of the username on the system then comparing the password supplied on the form with the password maintained for the account on the system.

On a modern Linux system, Pluggable Authentication Modules, or PAM, supports user PAM supported libraries and applications are available on the Internet to be used as a template in constructing a custom authentication package.

For the UMS, a web form will request a user to input his username and password. PAM will then authenticate this user. If the user is successfully authenticated, a session token will be created for the user. This token will have an expiry date one hour into the future.

Authenticating GPFN volunteers is a slightly different matter. A GPFN volunteer is not simply a user, but rather a user with increased permissions to the system this system may not be suitable for volunteers. As well, volunteers may not want to use their system password to authenticate themselves to the UMS. Instead, they may want to use a password specific to the UMS.

For this reason, authenticating GPFN volunteers will use the `htaccess` and `htpasswd` functions inherent in the Apache web server. Web pages and CGI scripts specific to volunteers will be in a separate directory under which the web server will apply access controls.

Session Credentials

Once the system authenticates a user, the libraries will have to manage session persistence for the user. To effect this persistence, the libraries will have their own database in the RDBMS. When a user is authenticated, a session token will be created composed of the username, the access level, the session creation timestamp, and a unique pseudo-random session ID. This token is inserted into the database and a cookie with this session token will be sent back to the user's browser.

From then on, any calls by the user to a page on the system will return the cookie. This cookie is matched with the cookie in the database. If a match exists, then the user is the holder of a currently valid session. Any of the following conditions indicates that the session is not currently valid:

1. There is no corresponding session token in the database matching the session token contained in the supplied cookie.
2. There is no cookie containing a session token sent back by the web browser.

3. The session tokens for this user between the database and the sent cookie do not match.
4. If the timestamp of the cookie as recorded in the database exceeds the current timestamp less the expiry period of the session.

In any of these cases, the user is redirected back to the login page with the current pages' URL stored in the QUERY_STRING for re-authentication and the session token is deleted from the database and the cookie. Once re-authenticated, the stored URL can be used to return the user to the page they were viewing before the authentication failure.

System Configuration APIs

All System Configuration APIs must be intrinsically distrustful of their calling functions. This prevents a potential malefactor from directly calling these programs while masquerading as coming from a valid user. Such a masquerade could be successful if the API did not verify the authenticity of the caller on its own.

This distrust is managed by creating a virtual domain within the webserver for the UMS. The virtual domain along with its libraries and system configuration APIs will run under a distinct set of user and group permissions.

Password Change

The Password Change API is responsible to ensure that the username and old password successfully authenticate the user. Only if the user is verified in this way is the new password checked for hardness according to the system configuration (normally using the cracklib libraries), then using a suid program, this API effects the password change on the system.

Mail Forwarding

The Mail Forwarding API authenticates the user and if authenticated, allows the user to review, change or set the contents of their home .forward file.

Mail Filtering

The Mail Filtering API can perform many functions. For the purpose of this application, it copies a system .procmairc template file into the user's home account, thus effecting spam control. Again, the API must authenticate the user before effecting these changes.

Reviewing Disk Quota

The Disk Quota API authenticates the user and once authenticated, returns the user's current quota values for display back to the user via a CGI script.

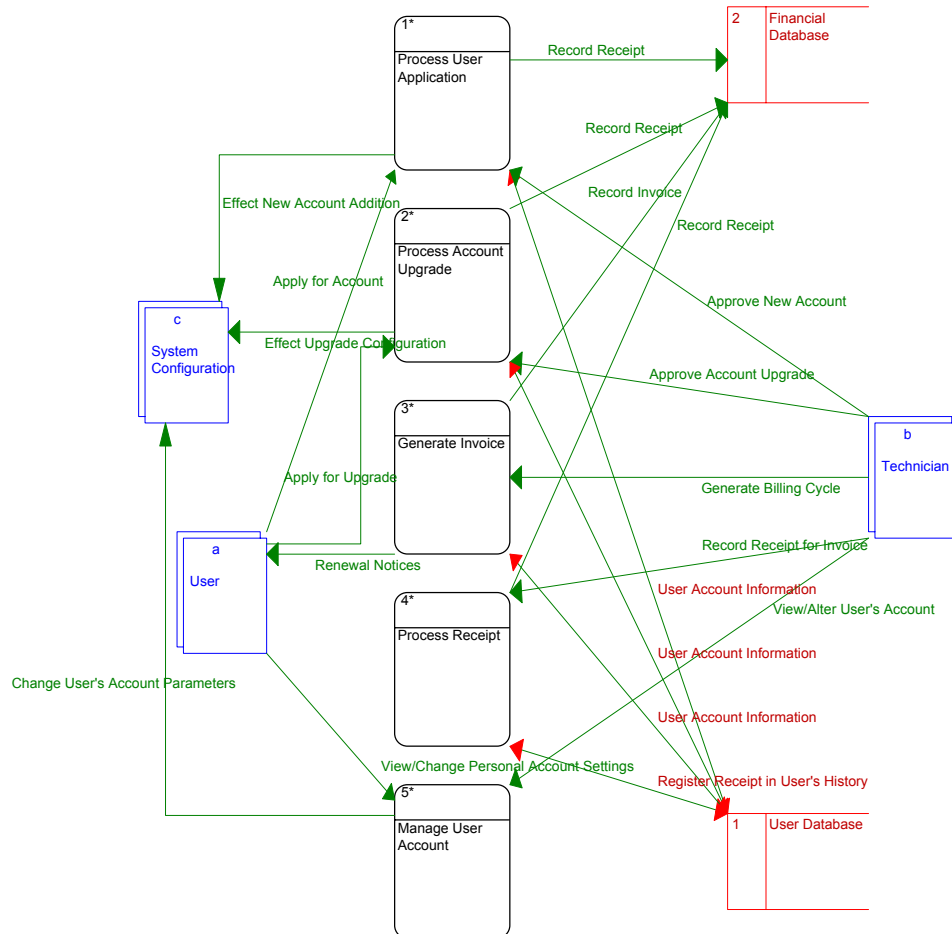
Data Flow Analysis

The system implementation involves a number of distinct yet inter-related functions. The following dataflow diagrams show the interaction flows in the system. The last dataflow case: "Manage User Account" is slated for Phases 2 and 3 of the project.

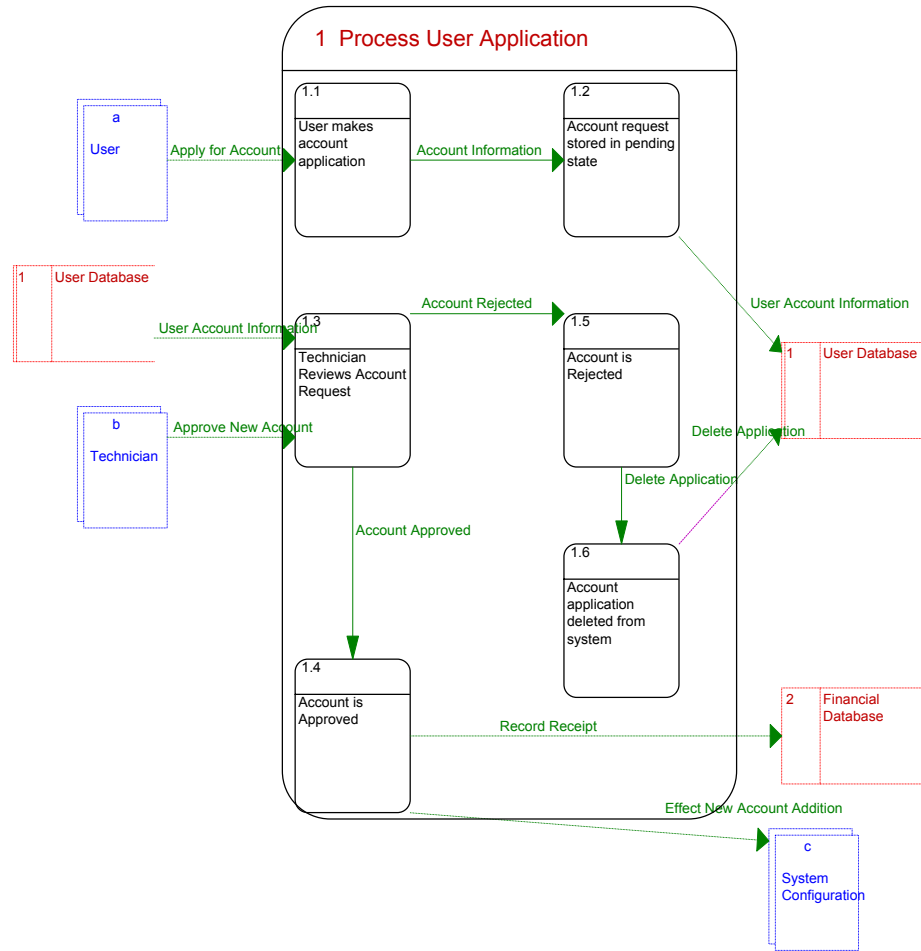
Data Flow Diagrams

Top Level Data Flow Diagram

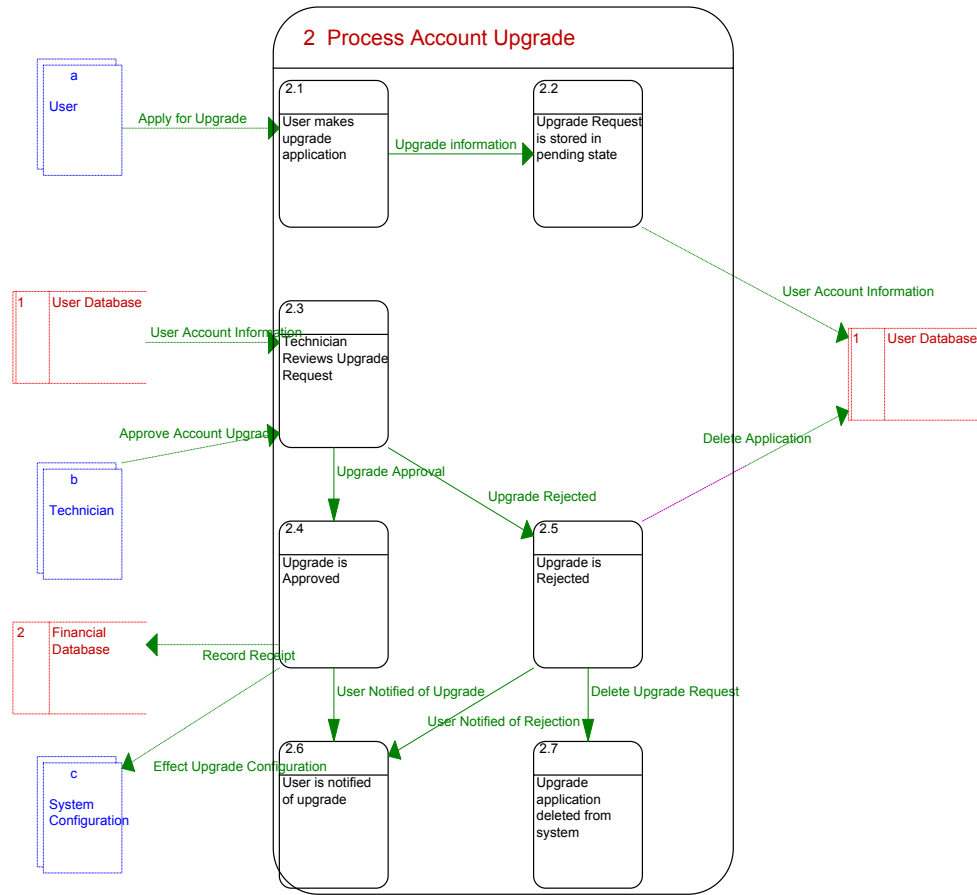
The top-level dataflow diagram lists the interactions between the system. This interaction is further exploded for each dataflow in the subsequent diagrams



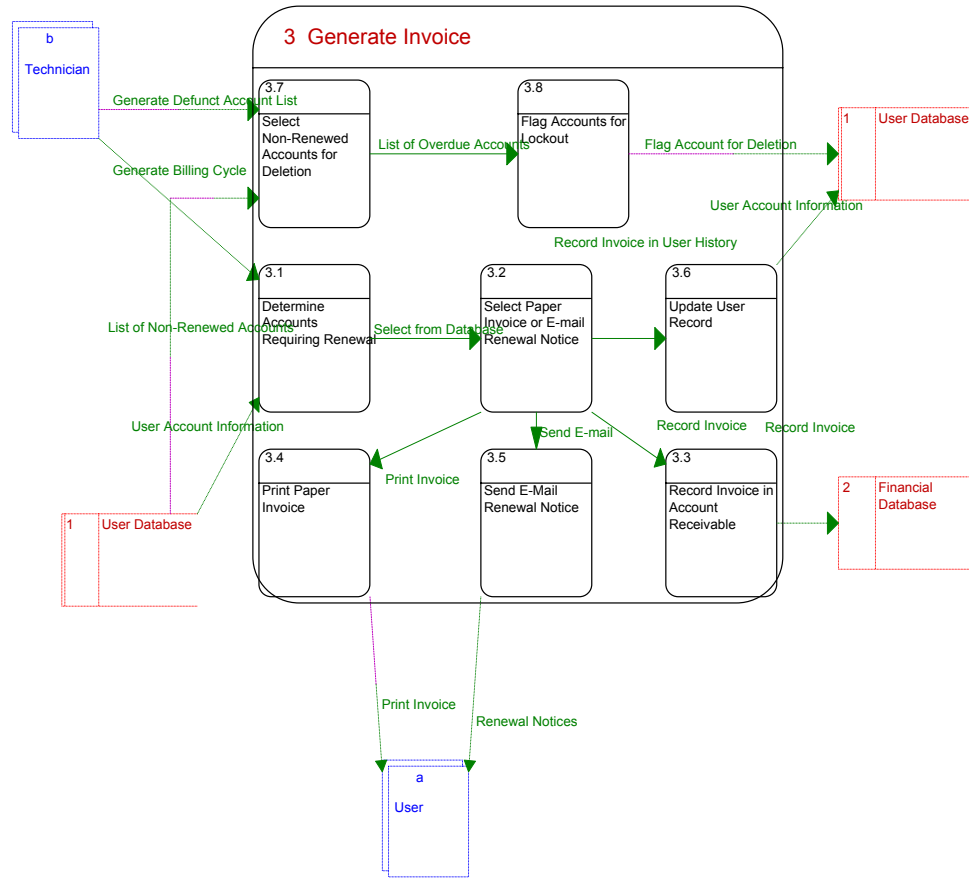
1. Process User Application Data Flow Diagram



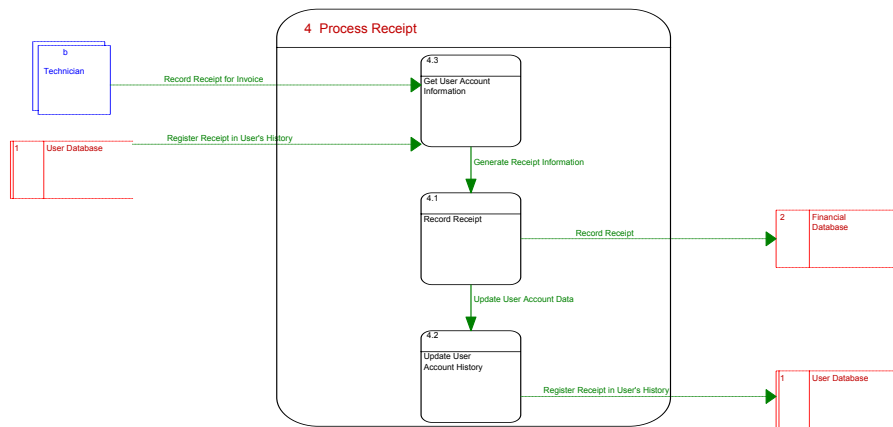
2. Process User Upgrade Data Flow Diagram



3. Generate Invoice Data Flow Diagram

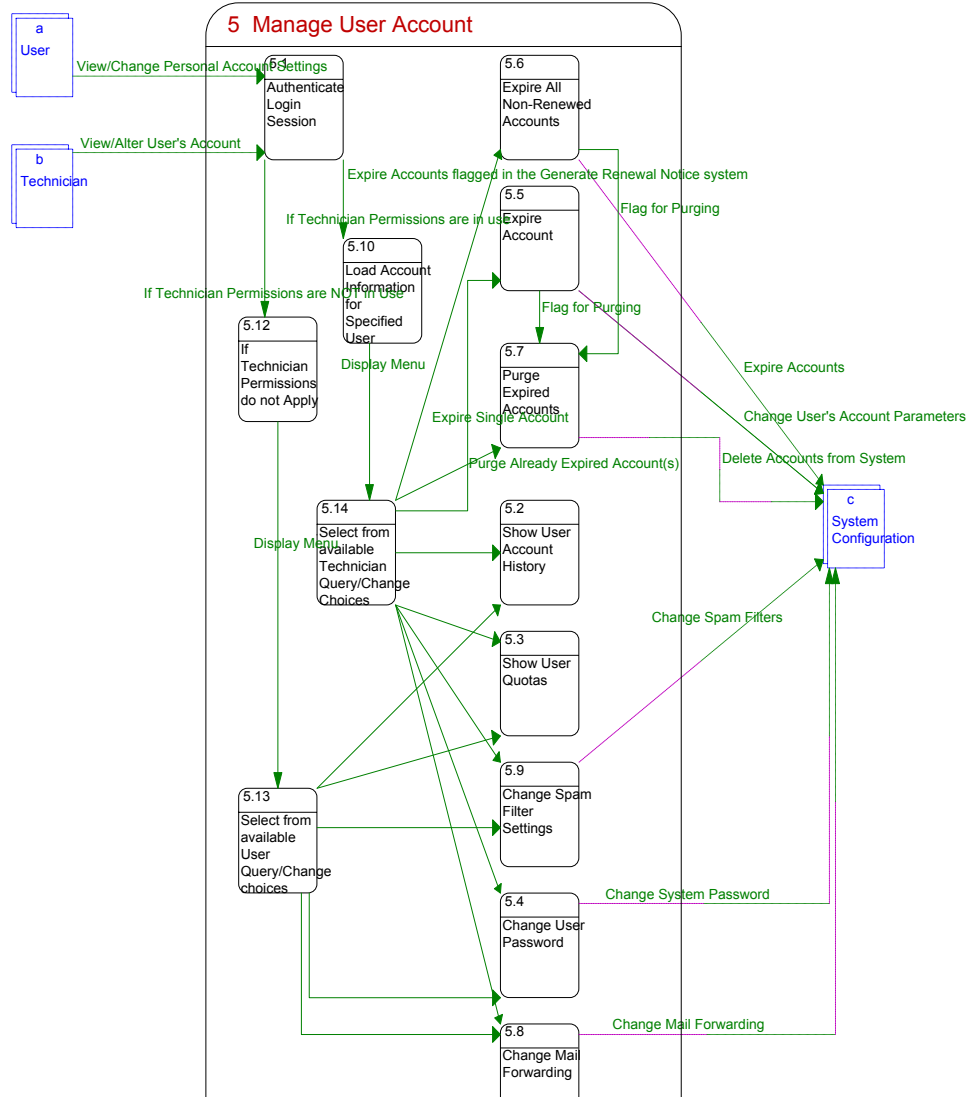


4. Process Receipt Data Flow Diagram



5. Manage User Account Data Flow Diagram

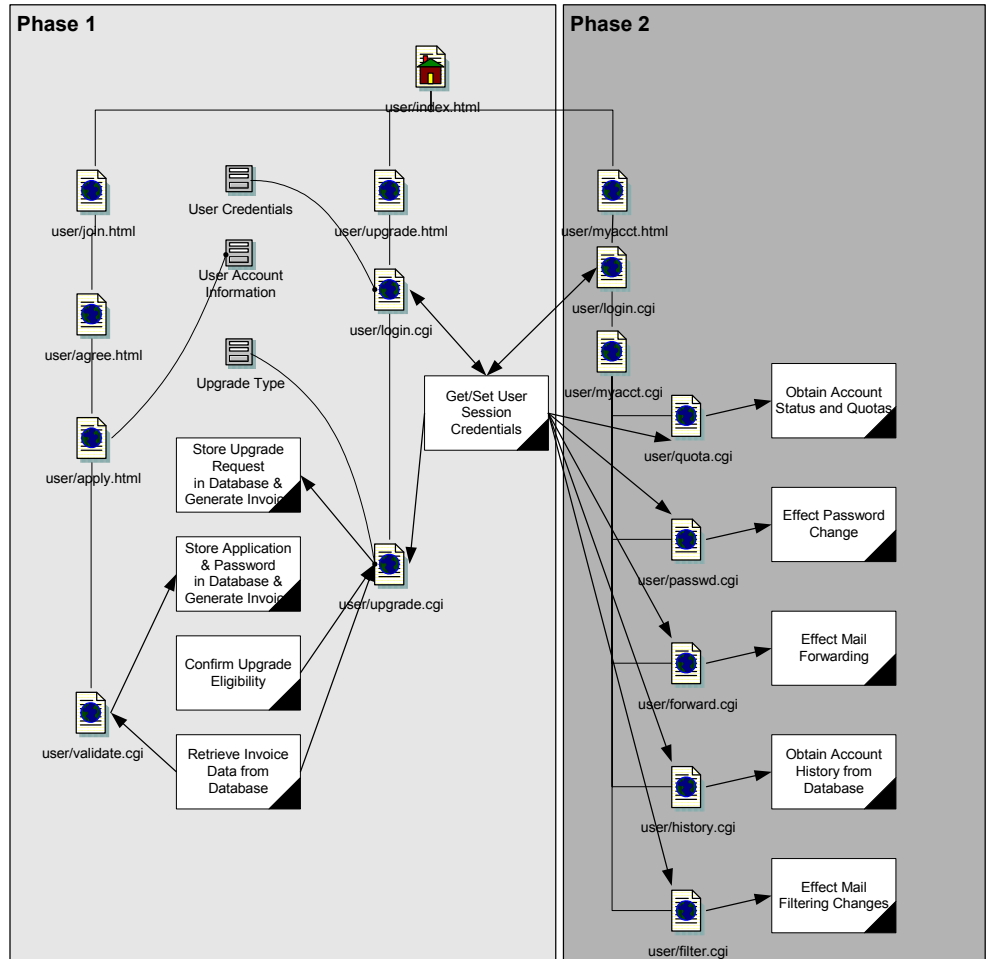
The Manage User Account Data Flow is slated for implementation in phases 2 and 3 of the project.



Web Navigation Model

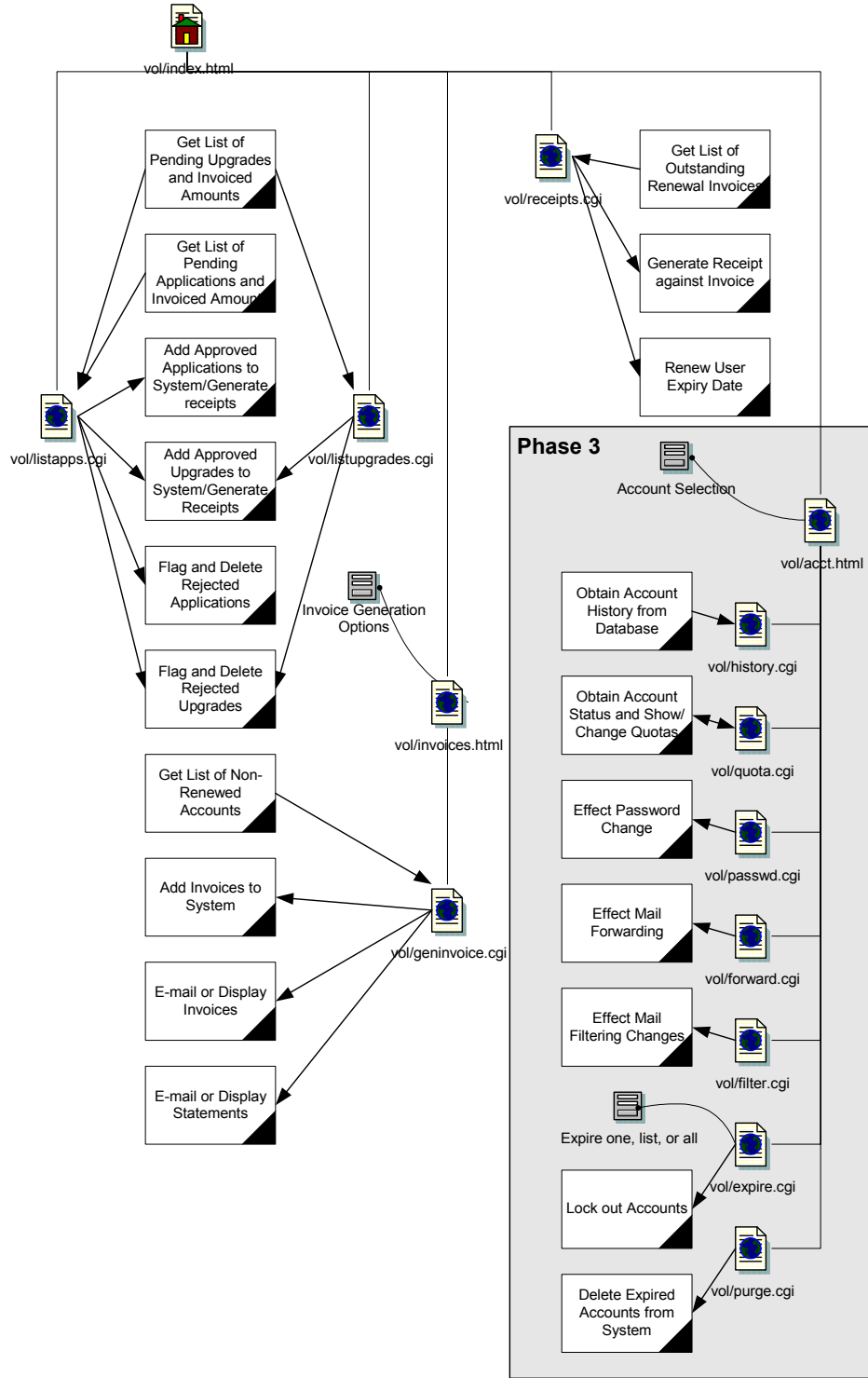
The dataflow will be managed using a combination of HTML pages, Javascript functions within HTML, CGI scripts, embedded PERL within HTML, and some shell script and PERL script library functions.

To visualize the data flow as it relates to the actual structure of the web site, the following Web Navigation Flow Diagrams were prepared. Figure 2 shows the pages available through the public web space to all users and potential users. Note that while an initial applicant can make an application without any sort of authentication, a user requesting an upgrade and all other user functions require the user to be authenticated prior to performing any of these actions.



• Figure 2: User Interactions with System

The volunteer portion of the web site will be secured using the inherent access controls of the Apache web server. As such, no user authentication library calls are required, the processing flow is somewhat simplified.

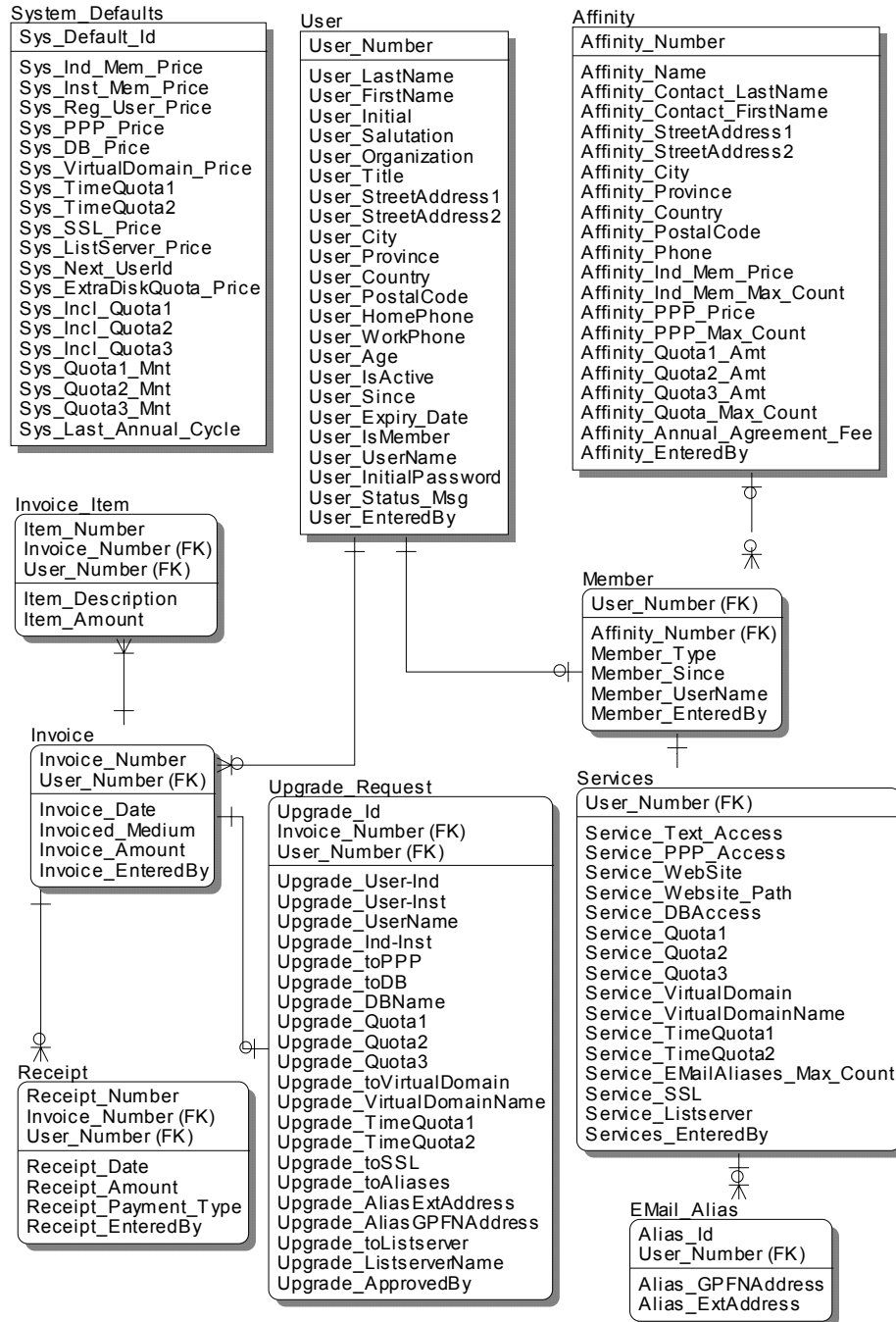


• Figure 3: GPFN Volunteer Interactions with System

Data Model

The Data Model is intended to be a comprehensive model to support both the immediate needs of the UMS and possible future features and enhancements. Not all fields will be used in the currently scoped implementation.

Entity Relationship Diagram



Data Dictionary

System_Default Table

Field Name	Data Type	Restrains	Default Value	Description
Sys_Default_Id	Int	Primary Key		System Generated Sequential Key
Sys_Ind_Mem_Price	Float	Not NULL	24.00	Approved Annual Fee for Individual Memberships
Sys_Inst_Mem_Price	Float	Not NULL	48.00	Approved Annual Fee for Institutional Memberships
Sys_Reg_User_Price	Float	Not NULL	0.00	Approved Annual Fee for Registered Non-member users
Sys_PPP_Price	Float	Not NULL	125.00	Approved Annual Fee for Graphical PPP Access
Sys_DB_Price	Float	Not NULL		Approved Annual Fee for Database Hosting
Sys_VirtualDomain_Price	Float	Not NULL		Approved Annual Fee for Virtual Domain Hosting
Sys_TimeQuota1	Float	Not NULL	?	Maximum allowed time per single session for users
Sys_TimeQuota2	Float	Not NULL	?	Maximum allowed sessions per day for users
Sys_SSL_Price	Float	Not NULL		Approved Additional Annual Fee for SSL encrypted Virtual Domain Hosting
Sys_ListServer_Price	Float	Not NULL	100.00	Approved Annual Fee for Listserver Hosting
Sys_Next_UserId	String (5)	Not NULL		Next available UserName for Registered User accounts following the Csuite naming convention of aannn ¹
Sys_ExtraDiskQuota_Price	Float	Not NULL	?	Approved Annual Pricing per Mb of extra Disk Quota above the approved included amounts
Sys_Incl_Quota1	Int	Not NULL		Approved Amount of basic disk space in filesystem1
Sys_Incl_Quota2	Int	Not NULL		Approved Amount of basic disk space in filesystem2
Sys_Incl_Quota3	Int	Not NULL		Approved Amount of basic disk space in filesystem3
Sys_Quota1_Mnt	String (8)	Not NULL		Mount point on the system for quota1
Sys_Quota2_Mnt	String (8)	Not NULL		Mount point on the system for quota2
Sys_Quota3_Mnt	String (8)	Not NULL		Mount point on the system for quota3
Sys_Last_Annual_Cycle	Time-stamp			The timestamp of the last generated invoice cycle. This is used to frequent duplicate invoice cycles within the same fiscal year.

¹ Csuite used a sequential numbering scheme where a new user was assigned a username of the form aannn where "a" denotes a lower case letter and "n" a number. Thus the first user to join the system was aa000, the second was aa001 and so on. User aa999 was followed by ab000 and so on.

User Table

Field Name	Data Type	Restrains	Default Value	Description
User_Number	Int	Primary Key		System Generated Sequential Number
User_LastName	String (30)	Not NULL		User's Last Name
User_FirstName	String (20)	Not NULL		User's First Name
User_Initial	String (60)			User's Initial
User_Salutation	String (60)		One of "Mr.", "Mrs.", "Ms.", "Dr.", "Rev."	User's Title of Address
User_Organization	String (50)			Name of User's Organization if user is a group or collective entity
User_Title	String (30)			Position or title of person within the organization
User_StreetAddress1	String (30)	Not NULL		User's first line of mailing address
User_StreetAddress2	String (30)			
User_City	String (30)	Not NULL		
User_Province	String (30)	Not NULL		
User_Country	String (30)	Not NULL		
User_PostalCode	String (20)	Not NULL		
User_HomePhone	String (20)			
User_WorkPhone	String (20)			
User_Age	Int			Really only required for users under the age of majority (so we get their parent's signature on any application)
User_IsActive	Int	Not NULL	0	0 for not active, 1 for active, 2 for declined, 3 for expired, 4 for purged
User_Since	Time-stamp	Not NULL		Date the user applied for an account
User_IsMember	Int	Not NULL	0	0 for non-member, 1 for member
User_Status_Msg	String (60)			Any Volunteer entered text string to describe issues concerning a user
User_UserName	String (16)	Not NULL		The system generated username for non- members, or the personalized username for members. This is only used during the account application process and NOT changed for users who upgrade to membership later.
User_InitialPassword	String (16)	Not NULL	Must be checked against the cracklib	The User requested initial password in MD5 hash encryption
User_EnteredBy	String (16)	Not NULL		The name of the GPFN Volunteer adding the user to the system

Member Table

Field Name	Data Type	Restrains	Default Value	Description
User_Number	Int	Foreign Key	References User	If an entry exists in this table with User_Number, then we know that this User is also a member
Affinity_Number	String (10)	Foreign Key	References Affinity	If an entry exists in this table, then the member has a special pricing relationship with GPFN (to be implemented in Phase 4)
Member_Type	Int	Not NULL	1 or 2	1 for Individual Member, 2 for Institutional Member
Member_Since	Time-stamp	Not NULL		Date on which user applied to become a member
Member_EnteredBy	String (16)	Not NULL		The name of the GPFN Volunteer adding the member to the system

Affinity Table (Phase 4)

Field Name	Data Type	Restrains	Default Value	Description
Affinity_Number	String (10)	Primary Key		
Affinity_Name	String (50)	Not NULL		Name of group or organization entering into an affinity relationship with GPFN
Affinity_Contact_LastName	String (30)	Not NULL		Name of contact person with the group or organization
Affinity_Contact_FirstName	String (20)	Not NULL		
Affinity_StreetAddress1	String (30)	Not NULL		
Affinity_StreetAddress2	String (30)			
Affinity_City	String (30)	Not NULL		
Affinity_Province	String (30)	Not NULL		
Affinity_Country	String (30)	Not NULL		
Affinity_PostalCode	String (30)	Not NULL		
Affinity_Phone	String (20)	Not NULL		
Affinity_Ind_Mem_Price	Float	Not NULL	0.00	Agreed price of individual memberships under this affinity agreement
Affinity_Ind_Mem_Max_Count	Int	Not NULL	0	Agreed maximum number of individual memberships to be sold under this affinity agreement
Affinity_PPP_Price	Float	Not NULL	0.00	Agreed price of PPP connections under this affinity agreement
Affinity_PPP_Max_Count	Int	Not NULL	0	Agreed maximum number of PPP connections to be sold under this affinity agreement

Affinity_Quota1_Amt	Int	Not NULL	0	Agreed amount of disk space in Quota space 1
Affinity_Quota2_Amt	Int	Not NULL	0	Agreed amount of disk space in Quota space 2
Affinity_Quota2_Amt	Int	Not NULL	0	Agreed amount of disk space in Quota space 2
Affinity_Quota_Max_Count	Int	Not NULL	0	Agreed maximum number of members who benefit from the increased quota limits under this affinity
Affinity_Annual_Agreement_Fee	Float	Not NULL	0.00	Annual Fee to be invoiced to the Affinity_Name for the terms of the affinity agreement
Affinity_EnteredBy	String (16)	Not NULL		The name of the GPFN Volunteer adding or editing the affinity to the system

Services Table

Field Name	Data Type	Restrains	Default Value	Description
User_Number	Int	Foreign Key	References User	
Service_Text_Access	Int	Not NULL	1	1 if user has access to text-only dial-up
Service_PPP_Access	Int	Not NULL	0	1 if user has access to PPP graphical dial-up
Service_WebSite	Int	Not NULL	0	1 if user has access to web space within GPFN's public web space (not including the user's public_html directory)
Service_Website_Path	String			The relative path from the Webserver's document root to the directory for this members' web space within the gpfn.ca domain.
Service_DBAccess	Int	Not NULL	0	Number of databases the user has access to on the database server
Service_Quota1	Int	Not NULL	0	Number of extra Mb of disk space the user has subscribed to on the filesystem referenced by Quota1
Service_Quota2	Int	Not NULL	0	Number of extra Mb of disk space the user has subscribed to on the filesystem referenced by Quota2
Service_Quota3	Int	Not NULL	0	Number of extra Mb of disk space the user has subscribed to on the filesystem referenced by Quota3
Service_VirtualDomain	Int	Not NULL	0	Number of virtual domains the user has on the system
Service_VirtualDomainName	String			Registered Name of the Virtual Domain
Service_TimeQuota1	Int	NotNULL	0	Amount of extra time the user has paid to be added to the value included in Sys_TimeQuota1
Service_TimeQuota2	Int	NotNULL	0	Amount of extra time the user has paid to be added to the value included in Sys_TimeQuota2
Service_EmailAliases_Max_Count	Int	Not NULL	0	The maximum number of e-mail aliases belonging to this user
Service_SSL	Int	Not NULL	0	The number of SSL enabled Virtual Domain web hosts for this

				user on the system
Service_Listserver	Int	Not NULL	0	The number of Listservers for this user on the system
Services_EnteredBy	String (16)	Not NULL		The name of the GPFN Volunteer adding or editing this service list on the system

Email_Alias Table

Field Name	Data Type	Restrains	Default Value	Description
Alias_Id	Int	Primary Key		System Generated Incremental Number
User_Number	Int	Foreign Key	References User	
Alias_GPFNAddress	String (30)	Not NULL		The GPFN address to which mail is received
Alias_ExtAddress	String (50)	Not NULL		The non-GPFN address to which mail is redirected

Invoice Table

Field Name	Data Type	Restrains	Default Value	Description
Invoice_Number	Int	Primary Key		System Generated Incremental Number
User_Number	Int	Foreign Key	References User	
Invoice_Date	Time-stamp	Not NULL		Date on which the invoice was issued
Invoice_Medium	String (20)	Not NULL		One of "e-mail", "paper", "fax" or some other such descriptor of the method in which the invoice was delivered
Invoice_Amount	Float	Not NULL		The total value of the invoice
Invoice_EnteredBy	String (16)	Not NULL		The name of the GPFN Volunteer adding this invoice to the system

Invoice_Item Table

Field Name	Data Type	Restrains	Default Value	Description
Item_Number	Int	Primary Key		System Generated Incremental Number
Invoice_Number	Int	Foreign Key	References Invoice	
User_Number	Int	Foreign Key	References User	
Item_Description	String (60)	Not NULL		Description of the line item for the invoice
Item_Amount	Float	Not NULL	0.00	Amount of the line item for the invoice

Receipt Table

Field Name	Data Type	Restrains	Default Value	Description
Receipt_Number	Int	Primary Key		System Generated Incremental Number
Invoice_Number	Int	Foreign Key	References Invoice	
User_Number	Int	Foreign Key	References User	
Receipt_Date	Time-stamp	Not NULL		Date that the amount was received and processed by the volunteer
Receipt_Amount	Float	Not NULL	0.00	Amount received from the user
Receipt_Payment_Type	String (30)	Not NULL		One of "Cash", "Cheque", "Credit Card" or some other description
Receipt_Entered_By	String (16)	Not NULL		The name of the GPFN Volunteer adding this receipt to the system

Upgrade_Request Table

Field Name	Data Type	Restrains	Default Value	Description
Upgrade_Id	Int	Primary Key		System Generated Incremental Number
Invoice_Number	Int	Foreign Key	References Invoice	
User_Number	Int	Foreign Key	References User	
Upgrade_User-Ind	Int	Not NULL	0	1 if the upgrade turns a registered user into an individual member
Upgrade_User-Inst	Int	Not NULL	0	1 if the upgrade turns a registered user into an institutional member
Upgrade_Ind-Inst	Int	Not NULL	0	1 if the upgrade turns an individual into an institutional member
Upgrade_Username	String (16)			The selected username for a user upgrading to a membership from a registered user account.
Upgrade_toPPP	Int	Not NULL	0	1 if the upgrade enables PPP
Upgrade_toDB	Int	Not NULL	0	The number of Databases to be added to the User's account
Upgrade_DBName	String			The requested name for a database on the database server.
Upgrade_Quota1	Int	Not NULL	0	Number of Mb to be added to the user's disk quota of the filesystem referenced by Quota1
Upgrade_Quota2	Int	Not NULL	0	Number of Mb to be added to the user's disk quota of the filesystem referenced by Quota2
Upgrade_Quota3	Int	Not NULL	0	Number of Mb to be added to the user's disk quota of the filesystem referenced by Quota3
Upgrade_toVirtualDomain	Int	Not NULL	0	Number of Virtual Domains to add to the user
Upgrade_VirtualDomainName	String			The requested domain for an application for a virtual domain
Upgrade_TimeQuota1	Int	Not NULL	0	Number of additional minutes to add to Sys_TimeQuota1
Upgrade_TimeQuota2	Int	Not NULL	0	Number of additional sessions to be added to Sys_TimeQuota2 for

				this user
Upgrade_toSSL	Int	Not NULL	0	Number of SSL enabled web hosts to be added to this client
Upgrade_toAliases	Int	Not NULL	0	Number of additional email aliases to be added to this user's services
Upgrade_AliasExtAddress	String			e-mail address of the external service to which e-mail to the alias will be redirected
Upgrade_AliasGPFNAddress	String			e-mail address of the GPFN address to which mail will be receive and then redirected
Upgrade_toListserver	Int	Not NULL	0	Number of additional Listservers to be added to this user's services
Upgrade_ListserverName	String			Name of mailing list for an upgrade to Listserver.
Upgrade_ApprovedBy	String (16)	Not NULL		The name of the GPFN Volunteer approving this upgrade on the system

Procedure Model

Web Pages

user/index.html

Procedure Model Type: <input checked="" type="checkbox"/> Web Page () CGI Script () Shared Library () System API Name: user/index.html Assigned to: Reference:	
Description	This pure HTML page contains the descriptions and navigation for users, prospective users and members to interact with the UMS.
Implementation Skills	HTML
Parameter List	None
Called By:	GPFN Main page, other pages on the GPFN web site
Can Call:	user/join.html user/upgrade.html user/myacct.html (Phase 2)
Function Description	Page invites viewer to join the GPFN listing the benefits of membership and other service upgrade options. (Phase 2)A link to the myacct.html page so users can view their account information is also present.
Possible Exit Conditions and Return Values	None required
Sign Off by:	Membership Committee or GPFN Board Designate

user/join.html

Procedure Model	
Type: <input checked="" type="checkbox"/> Web Page () CGI Script () Shared Library () System API	
Name: user/join.html	
Assigned to:	
Reference:	
Description	This HTML page contains the descriptions of the membership and user classes of the Great Plains Free-Net for which they wish to apply.
Implementation Skills	HTML
Parameter List	None
Called By:	user/index.html
Can Call:	user/agree.html user/index.html /index.html
Function Description	The page describes the three classes of user: Registered User, Individual Member and Institutional Member and the benefits and features of each class. A Cancel link redirects users to the user/index.html page and a GPFN button takes the viewer back to the main GPFN page.
Possible Exit Conditions and Return Values	None
Sign Off by:	Membership Committee or GPFN Board Designate

user/agree.html

Procedure Model	
Type: <input checked="" type="checkbox"/> Web Page () CGI Script () Shared Library () System API	
Name: user/agree.html	
Assigned to:	
Reference:	
Description	This page contains the user agreement, acceptable use policy and other requirements the viewer must agree to before obtaining a GPFN account.
Implementation Skills	HTML, HTML forms, Javascript
Parameter List	None
Called By:	user/join.html
Can Call:	user/apply.html user/index.html /index.html
Function Description	The page contains the text to which a prospective user must agree before an account application is received and processed. The viewer notes acceptance of this agreement by clicking an "Accept" button at the bottom of the page. The Accept button directs them to the apply.html page with the value "UA_accept" appended to the URL as a Query String parameter. A "Decline" button returns them to the user/join.html page. A "GPFN Home" button returns them to the GPFN main page.
Possible Exit Conditions and Return Values	If the viewer clicks the "Accept" button, they are sent to user/apply.html?UA_accept. If the viewer clicks the "Decline" button, they are sent to user/join.html If the viewer clicks the "GPFN Home" button, they are sent to /index.html
Sign Off by:	Membership Committee or GPFN Board Designate

user/apply.html

Procedure Model Type: <input checked="" type="checkbox"/> Web Page () CGI Script () Shared Library () System API Name: user/apply.html Assigned to: Reference:	
Description	This HTML page contains a form to gather the information required to process an application to the GPFN
Implementation Skills	HTML, HTML Forms, Javascript
Parameter List	Query String must contain value "UA_accept"
Called By:	user/agree.html
Can Call:	user/validate.cgi user/index.html user/join.html /index.html
Function Description	<p>The URL to which the user arrives must be "user/apply.html?UA_accept". If the "?UA_accept" portion of the URL is missing, they should be redirected to the user/join.html page.</p> <p>If the UA_accept parameter is received the page displays an HTML form. This form must first query the viewer for the type of account they want. A pop-up menu named "Membership_Class" should list "Registered User", "Individual Member" and "Institutional Member" and send values "Registered", "Individual", and "Institution" respectively.</p> <p>The form must also obtain the following data from the user. Items marked with a (*) are required and must be present before the form is processed:</p> <ul style="list-style-type: none"> *User_FirstName *User_LastName User_Initial *User_Salutation (popup one of Mr., Mrs., Ms., Dr., Rev.,) *User_Organization (only required for an Institutional Membership) *User_Title (only required for an Institutional Membership) *User_StreetAddress1 User_StreetAddress2 *User_City *User_Province *User_Country *User_PostalCode User_HomePhone User_WorkPhone *User_Age (If under 18 and needing a parent's approval for an account) *User_InitialPassword (entered as a password field of 6-8 characters. Users should be told to only use letters, numbers, and underscores and to NOT use any real word variation of a real word, any variation of their name or username) <p>If the application is for an individual or institutional membership the following fields must also be gathered</p> <ul style="list-style-type: none"> Upgrade_UserName (the requested username by the prospective member) Upgrade_toPPP (check box: True="1" default: False='0') Upgrade_toDB (check box: True="1" default: False='0') Upgrade_DBName (textfield—only meaningful if Upgrade_toDB is checked) Upgrade_toVirtualDomain (check box: True="1" default: False='0') Upgrade_VirtualDomainName (textfield—only meaningful if Upgrade_toVirtualDomain is checked) Upgrade_toSSL (check box: True="1" default: False='0') Upgrade_toListserver (check box: True="1" default: False='0') Upgrade_ListserverName (textfield—only meaningful if Upgrade_toListserver is checked) Upgrade_toAliases (user enters a number default: False='0') Upgrade_AliasExtAddress (textfield—only meaningful if Upgrade_toAlias is checked)

	Upgrade_AliasGPFNAddress (textfield—only meaningful if Upgrade_toAlias is checked) Upgrade_Quota1 (check box: True="5" default: False=0) Upgrade_Quota2 (check box: True="5" default: False=0) Upgrade_Quota3 (check box: True="5" default: False=0) All requests for upgrades should list the annual (and possibly with Javascript the pro-rated) amounts of the upgrade.
Possible Exit Conditions and Return Values	<ol style="list-style-type: none"> 1. If UA_accept parameter is missing, user is redirected back to user/join.html 2. If any of the required fields listed above are left blank, the submit button should be disabled or the user should be directed to fill in these fields before proceeding.
Sign Off by:	Membership Committee or GPFN Board Designate

user/upgrade.html

Procedure Model	
Type:	<input checked="" type="checkbox"/> Web Page () CGI Script () Shared Library () System API
Name:	user/upgrade.html
Assigned to:	
Reference:	
Description	This HTML page contains the terms and conditions related to any upgrade option including prices.
Implementation Skills	HTML
Parameter List	None
Called By:	user/index.html
Can Call:	user/login.cgi?upgrade.cgi user/index.html /index.html
Function Description	The page lists all relevant details for users to be able to decide what options to select in upgrading their account. Users can click to move forward to the login.cgi page before formally request an upgrade, or click a cancel button to go back the user/index.html page or a GPFN home button.
Possible Exit Conditions and Return Values	If the user clicks to apply for an upgrade, control passes to URL: login.cgi?upgrade.cgi. The "upgrade.cgi" string allows the login.cgi script to redirect the user to the correct option page if the login is successful.
Sign Off by:	Membership Committee or GPFN Board Designate

user/myacct.html

Procedure Model	
Type: <input checked="" type="checkbox"/> Web Page () CGI Script () Shared Library () System API	
Name: user/myacct.html (Phase 2)	
Assigned to: Phase 2	
Reference:	
Description	This HTML page contains a description of the options available to the account holder in viewing or managing his or her account history and configuration
Implementation Skills	HTML
Parameter List	None
Called By:	user/index.html
Can Call:	user/login.cgi?myacct.cgi user/index.html /index.html
Function Description	Users can click to move forward to the login.cgi page before formally viewing their account history or settings or altering their account configuration, or click a cancel button to go back the user/index.html page or a GPFN home button.
Possible Exit Conditions and Return Values	If the user clicks to view their account, control passes to the login.cgi?myacct.cgi. The "myacct.cgi" string is used to allow the login.cgi script to redirect the user to the correct option page if the login is successful.
Sign Off by:	Membership Committee or GPFN Board Designate

vol/index.html

Procedure Model	
Type: <input checked="" type="checkbox"/> Web Page () CGI Script () Shared Library () System API	
Name: vol/index.html	
Assigned to:	
Reference:	
Description	This pure HTML page allows the volunteer to choose between processing applications for accounts and upgrades and processing financial transactions.
Implementation Skills	HTML
Parameter List	None
Called By:	Directly by the user. Page is behind htaccess controls.
Can Call:	vol/listapps.cgi vol/upgrades.cgi vol/receipts.cgi vol/invoices.html vol/acct.html (Phase 3)
Function Description	The volunteer can choose: <ul style="list-style-type: none"> • a link to listapps.cgi to get a list of all new account applications for processing; • a link to upgrades.cgi to get a list of all upgrade requests to existing accounts; • a link to receipts.cgi to process a receipt to a renewal invoice; • a link to invoices.html to select the options before creating a new invoice cycle or sending statements to accounts; • a link to acct.html to get a list of available account management options so the volunteer can view or alter the user's account history or settings

	(Phase 3)
Possible Exit Conditions and Return Values	None
Sign Off by:	Membership Committee or GPFN Board Designate

vol/invoices.html

Procedure Model	
Type: <input checked="" type="checkbox"/> Web Page () CGI Script () Shared Library () System API	
Name: vol/invoices.html	
Assigned to:	
Reference:	
Description	<p>This HTML/Javascript page allows the volunteer to choose the options before generating new invoices or reminder notices. The volunteer can choose to generate an annual invoice cycle for all members, display an invoice for a particular user, generate an annual invoice for a particular user, display statements for all members, or display a statement for a particular user.</p> <p>As well, the form allows the volunteer to choose to e-mail the statements or invoices to the user, or to print them to the screen, or to print them to a tab delimited file which can be downloaded. The number and identity of the invoices or statements e-mailed or displayed in this way corresponds to the choices made in the preceding paragraph.</p>
Implementation Skills	HTML, Javascript, HTML forms
Parameter List	None
Called By:	vol/index.html
Can Call:	vol/geninvoice.cgi vol/index.html
Function Description	<p>The HTML form gathers the following information before passing the data to the geninvoice.cgi script:</p> <ol style="list-style-type: none"> *Invoice_Count (Radio Button set to "1" or "all") User_UserName (Textfield matching the data definition entry and must be filled in with the UserName of the User for whom the invoice/statement is to be created only if the Invoice_Count is set to "1") *Type_Switch (Radio Button set to "NewInvoice", "ExistingInvoice" or "Statement". These values will have English equivalences such as "Generate New User Invoices", "View an Existing User Invoice", or "Print User Statements". Javascript should implement the rule that if Invoice_Count is set to 1, then the NewInvoice option is NOT available. *Medium_Switch (Radio Button set to "E-Mail" or "Screen") <p>A Submit button forwards the form data to the geninvoice.cgi. A Cancel button returns the volunteer to the vol/index.cgi page.</p>
Possible Exit Conditions and Return Values	The form and Javascript should correctly handle all data exceptions. Data should be passed to the geninvoice.cgi script. Cancels should return control to index.html
Sign Off by:	Membership Committee or GPFN Board Designate

vol/invoice.tmpl

Procedure Model	
Type: <input checked="" type="checkbox"/> Web Page () CGI Script () Shared Library () System API	
Name: vol/invoice.tmpl	
Assigned to:	
Reference:	
Description	This HTML page forms a dynamically generated invoice. The template file is used by the geninvoice.cgi developer to present an attractive and visually consistent invoice to the user.
Implementation Skills	HTML
Parameter List	None
Called By:	vol/geninvoice.cgi
Can Call:	
Function Description	The template file includes all HTML and graphics to display an invoice on a web browser window. It must be visually equivalent whether displayed in Internet Explorer, Netscape Navigator or Opera. Spaces should be left for user identification, date of invoice, invoice charges and item descriptions, payment terms and options, instructions to print this page, and GPFN identification and remittance information
Possible Exit Conditions and Return Values	The page is not called directly but will be cut and pasted into cgi scripts.
Sign Off by:	Membership Committee or GPFN Board Designate

vol/stmt.tmpl

Procedure Model	
Type: <input checked="" type="checkbox"/> Web Page () CGI Script () Shared Library () System API	
Name: vol/stmt.tmpl	
Assigned to:	
Reference:	
Description	This HTML page forms a dynamically generated statement. The template file is used by the geninvoice.cgi developer to present an attractive and visually consistent statement to the user.
Implementation Skills	HTML
Parameter List	None
Called By:	vol/geninvoice.cgi
Can Call:	
Function Description	The template file includes all HTML and graphics to display a statement on a web browser window. It must be visually equivalent whether displayed in Internet Explorer, Netscape Navigator or Opera. Spaces should be left for user identification, dates and amounts of invoices and receipts, payment terms and options, outstanding balance and the aging of the balance and GPFN identification, instructions to print this page, and remittance information
Possible Exit Conditions and Return Values	The page is not called directly but will be cut and pasted into cgi scripts.
Sign Off by:	Membership Committee or GPFN Board Designate

vol/rcpt.tmpl

Procedure Model	
Type: <input checked="" type="checkbox"/> Web Page () CGI Script () Shared Library () System API	
Name: vol/rcpt.tmpl	
Assigned to:	
Reference:	
Description	This HTML page forms a dynamically generated receipt. The template file is used by the receipts.cgi developer to present an attractive and visually consistent statement to the user.
Implementation Skills	HTML
Parameter List	None
Called By:	vol/receipts.cgi
Can Call:	
Function Description	The template file includes all HTML and graphics to display a receipt on a web browser window. It must be visually equivalent whether displayed in Internet Explorer, Netscape Navigator or Opera. Spaces should be left for user identification, date, instructions to print this page, amount and payment method as well as GPFN identification and a thank-you message.
Possible Exit Conditions and Return Values	The page is not called directly but will be cut and pasted into cgi scripts.
Sign Off by:	Membership Committee or GPFN Board Designate

vol/accept.tmpl

Procedure Model	
Type: <input checked="" type="checkbox"/> Web Page () CGI Script () Shared Library () System API	
Name: vol/accept.tmpl	
Assigned to:	
Reference:	
Description	This HTML page forms a dynamically generated statement. The template file is used by the validate.cgi developer to present a user acceptance agreement for signature and mail-in by the new account applicant.
Implementation Skills	HTML
Parameter List	None
Called By:	user/validate.cgi
Can Call:	
Function Description	The template file includes all HTML and graphics to display a user acceptance agreement on a web browser window. It must be visually equivalent whether displayed in Internet Explorer, Netscape Navigator or Opera. Spaces should be left for user identification, legal terms and obligations, provisions for under-age applicants, payment terms and options, and GPFN identification, instructions to print this page, and remittance information
Possible Exit Conditions and Return Values	The page is not called directly but will be cut and pasted into cgi scripts.
Sign Off by:	Membership Committee or GPFN Board Designate

vol/acct.html

Procedure Model	
Type: <input checked="" type="checkbox"/> Web Page () CGI Script () Shared Library () System API	
Name: vol/acct.html (Phase 3)	
Assigned to: Phase 3)	
Reference:	
Description	This HTML page lists the account management options available to the volunteer and collects the information on which user account to display.
Implementation Skills	HTML, HTML Forms
Parameter List	None
Called By:	vol/index.html
Can Call:	vol/history.cgi vol/quota.cgi vol/passwd.cgi vol/forward.cgi vol/filter.cgi vol/expire.cgi vol/purge.cgi
Function Description	Not yet fully defined
Possible Exit Conditions and Return Values	Not yet fully defined
Sign Off by:	Membership Committee or GPFN Board Designate

CGI Procedures

user/validate.cgi

Procedure Model Type: ()Web Page (<u>X</u>)CGI Script ()Shared Library ()System API Name: user/validate.cgi Assigned to: Reference:	
Description	This script takes the information from the user/apply.html form, verifies that all required data is present before attempting to create a new user request.
Implementation Skills	PERL, CGI, SQL
Parameter List	Membership_Class User_FirstName User_LastName User_Initial User_Salutation User_Organization User_Title User_StreetAddress1 User_StreetAddress2 User_City User_Province User_Country User_PostalCode User_HomePhone User_WorkPhone User_Age User_InitialPassword Member_UserName (the requested username by the prospective member) Upgrade_toPPP (check box: True="1") Upgrade_toDB (check box: True="1") Upgrade_toVirtualDomain (check box: True="1") Upgrade_toSSL (check box: True="1") Upgrade_toListserver (check box: True="1") Upgrade_toAliases (user enters a number) Upgrade_Quota1 (check box: True="5") Upgrade_Quota2 (check box: True="5") Upgrade_Quota3 (check box: True="5")
Called By:	user/apply.html
Can Call:	lib/uname_generate.pl lib/uname_test.pl lib/pw_check.pl
Function Description	<ol style="list-style-type: none"> 1. The script verifies that the user has supplied all required values. The procedure model for the user/apply.html page lists these requirements. They include: Membership_Class, User_FirstName, User_LastName, User_Salutation, User_StreetAddress1, User_City, User_Province, User_Country, User_PostalCode, User_Age, User_InitialPassword. 2. If Membership_Class is "Institution" or "Individual", the following additional fields must also be gathered: Member_UserName, Upgrade_toPPP, Upgrade_toDB, Upgrade_toVirtualDomain, Upgrade_toSSL, Upgrade_toListserver, Upgrade_toAliases, Upgrade_Quota1, Upgrade_Quota2, Upgrade_Quota3. 3. If Membership_Class is "Institution" User_Organization, and User_Title are also required fields. 4. If Membership_Class is "Registered", then the library function "uname_generate.pl" is called to return the next sequential userid. Otherwise, the "uname_test.pl" is called to verify that the value supplied in the Member_UserName field is unique and valid. If

	<p>uname_generate.pl or uname_test.pl returns an error, the script returns with error condition 1 below.</p> <ol style="list-style-type: none"> 5. The value of User_InitialPassword is tested with a call to "pwcheck.pl". If the test fails, the script returns with error condition 1 below. 6. The values from the form are inserted as a new record in the Database User table. In addition, the following additional fields are populated as follows: User_IsActive = "Pending" User_Since = current timestamp User_IsMember = "Y" if Membership_Class is "Individual" or "Institution", otherwise User_Is_Member = "N" User_EnteredBy = Web Server's 'REMOTE_USER' value Retrieve the value of User_Number from the new record 7. If Membership_Class is 'Registered', proceed to step 14 below. 8. If Membership_Class is not "Registered", insert a new record in the Member table with the following values: User_Number = the User_Number of the just created User record in the User table Member_Type = 'Inst' or 'Ind' depending on whether Membership_Class is 'Institution' or 'Individual' respectively Member_UserName = User_UserName Member_Since = User_Member_Since Member_EnteredBy = the Web Server's REMOTE_USER value 9. If Membership_Class is not 'Registered' and any of Upgrade_toPPP, Upgrade_toDB, Upgrade_toVirtualDomain, Upgrade_toSSL, Upgrade_toListserver, Upgrade_toAliases, Upgrade_Quota1, Upgrade_Quota2, Upgrade_Quota3 are True or not equal to '0', create a new record in the Upgrade_Request table inserting these values into the record. 10. Unless Membership_Class is 'Registered', insert a new record into the Invoice table with the following values: Invoice_Date is the current Timestamp Invoice_Medium = 'Web' Invoice_EnteredBy = REMOTE_USER and retrieve the Invoice_Number of this new record 11. Insert a new record in the Invoice_Item table with the current Invoice_Number and User_Number. If Membership_Class is 'Institution' then the Item_Description should say 'Institutional Annual Membership' and the Item_Amount is the value of the Sys_Inst_Mem_Price. Otherwise, the Item_Description should say 'Individual Annual Membership' and the Item_Amount is the value of the Sys_Ind_Mem_Price. 12. For each of the upgrade options listed in step 9 above where the option is not 0 or is "False", insert a new record in the Invoice_Item table with an appropriate description and the value of the option field multiplied by the corresponding value from the System Defaults table. 13. Calculate the total cost of the Invoice by taking the sum of all Invoice_Item records for this particular Invoice_Number and put this value in the Invoice_Amount field for the corresponding Invoice. 14. Display the User Acceptance agreement based on the contents of the accept.tpl page created by the web designers. Users will be instructed to print out this form and send it with any required documentation to the office. 15. If Membership_Class is not Registered, append the Invoice information to the User Acceptance agreement created above using the format supplied in the invoice.tpl file.
<p>Possible Exit Conditions and Return Values</p>	<ol style="list-style-type: none"> 1. The user supplied Member_UserName is not valid or not unique. An error message informing the user of the nature of the problem is displayed on the returned HTML page and the form in the user/apply.html page is recreated and repopulated with the current values. The user is invited to pick a different UserName and resubmit the form. 2. The user supplied Initial_Password is not valid or not sufficiently hard. An error message informing the user of the specific failure of the

	password is displayed on the returned HTML page and the form in the user/apply.html page is recreated and repopulated with the current values. The user is invited to pick a different InitialPassword and resubmit the form.
Sign Off by:	Project Manager & Membership Committee or Board Designate

user/login.cgi

Procedure Model	
Type: ()Web Page (<u>X</u>)CGI Script ()Shared Library ()System API	
Name: user/login.cgi	
Assigned to:	
Reference:	
Description	This script creates a form to which the user enters their username and password, validates the information against the system records and if the user is authenticated, sets up a set of credentials for the user's current session.
Implementation Skills	PERL, CGI, SQL
Parameter List	QueryString contains the URL (without the host portion) to which the user is redirected upon a successful authentication
Called By:	user/upgrade.html user/myacct.html
Can Call:	lib/Login.pm user/upgrade.cgi user/myacct.cgi user/index.html
Function Description	<ol style="list-style-type: none"> 1. The login.cgi script queries the user for a username and password combination. For users, the default access_level is "0".² 2. The username is then sent to the auth_user command. If the user is successfully validated according to the auth_user command, the Login.pm's constructor method is called and a Login object is returned. This Login object is used to create a new cookie called Login with the value "<username>:<access_level>:<session_token>" and a 1 hour expiry period. This cookie is returned to the user's browser. 3. If successful, the user is redirected to the URL constructed from the relative path and filename contained within the QueryString variable. Otherwise, the user is returned to the login.cgi page again with the appropriate error message displayed. The user is also given a link to go to the user/index.html page.
Possible Exit Conditions and Return Values	<ol style="list-style-type: none"> 1. A successful return from the auth_user command will redirect the user to the URL constructed from the relative path and filename passed through the QueryString variable (this value is relative in that it is missing the host and domain portion of a fully formed URL). 2. An unsuccessful return from the auth_user command should return the user to another invocation of the login.cgi script and display the relevant error message. 3. The user is always given the option of going to the user/index.html page or to the GPFN main page.
Sign Off by:	Project Manager

² The Access_Level value is included so that access texture can be incorporated in the future when different users and different volunteers can be given different levels of access to the system.

user/upgrade.cgi

Procedure Model Type: ()Web Page <u>(X)CGI Script</u> ()Shared Library ()System API Name: user/upgrade.cgi Assigned to: Reference:	
Description	This script shows the user the available system upgrades available to them. If the user selects any upgrades, an entry is added to the upgrade table so that the upgrades can be processed and applied to the users account.
Implementation Skills	PERL, CGI, SQL, Javascript
Parameter List	
Called By:	user/index.html
Can Call:	lib/Login.pm lib/uname_test.pl vol/invoice.tpl
Function Description	<ol style="list-style-type: none"> 1. The script calls the Login.pm module's getLogin method to see if a valid set of user credentials exist for this user. If getLogin returns a NULL value, the user is redirected to the user/index.html page. 2. If the getLogin returns a session credential, the Login.pm module's getUsername method is called to determine the username of the session's owner. 3. The list of available upgrades for the user is determined and displayed to the user using the following logic: 4. If the User record associated with the UserName has no corresponding entry in the Member table, then the Upgrade_User-Ind and Upgrade_User-Inst radio button is displayed. Only one of these options can be selected by the user. Javascript should be used to ensure that the user cannot select any other upgrade option unless one of these radio buttons is selected. 5. If the Upgrade_User-Ind or Upgrade_User-Inst radio button is selected, ask the user to enter a textfield of between 4 and 16 characters for the Upgrade_Username field. 6. If a corresponding entry in the Member table exists for this user, and the Member_Type is "Individual", a checkbox for Upgrade_ind-Inst is displayed. 7. If the corresponding entry in the Services table for this User_Number does not have a value for Service_PPP_Access, or Service_PPP_Access = 0, then a checkbox for Upgrade_toPPP is shown. 8. If the corresponding entry in the Member_Type is "Institution" or the checkbox "Upgrade_Ind-Inst" or the radio button "Upgrade_User-Inst" is selected above, and the corresponding entry in the Services table for this User_Number does not have a positive integer value in the Service_WebSite field, then display a checkbox for Upgrade_toWebSite and by default set it to "checked." 9. If there is a corresponding entry in the Member table for this User_Number, display the following checkboxes: Upgrade_toVirtualDomain Upgrade_toDB Upgrade_toListserver and collect the following information in a text field for each of these checked options: Upgrade_VirtualDomainName Upgrade_DBName Upgrade_ListserverName 10. If there is a corresponding entry in the Services table for Service_WebSite, or for Service_VirtualDomain, or the checkboxes for Upgrade_toVirtualDomain or Upgrade_toWebSite are selected above, display a checkbox for Upgrade_toSSL.

	<ol style="list-style-type: none"> 11. If the corresponding entry in the Member_Type is "Institution" or the checkbox "Upgrade_Ind-Inst" or the radio button "Upgrade_User-Inst" is selected above, then display a checkbox for Upgrade_toAliases include two text fields for "Upgrade_AliasExtAddress" and "Upgrade_Alias_GPFNAddress" 12. If there is a corresponding Member record for this User_Number, display three text fields with the current values for Service_Quota1, Service_Quota2, and Service_Quota3 for the corresponding Service table entry for this user. The labels for these fields should be constructed with meaningful labels from the Sys_Quota1_Mnt, Sys_Quota2_Mnt, and Sys_Quota3_Mnt fields. 13. If there is no corresponding Member record for this User_Number and the Upgrade_User-Ind or Upgrade_User-Inst checkboxes are checked above, then display the default quotas from the Sys_Incl_Quota1, Sys_Incl_Quota2, and Sys_Incl_Quota3 fields as text fields. 14. If a Username is requested in step 5 above, test this Username against the uname_test.pl library function. 15. Collect the information and store all values in the respective fields of a new Upgrade_Request record. 16. Insert a new record into the Invoice table with the following values: Invoice_Date is the current Timestamp Invoice_Medium = 'Web' Invoice_EnteredBy = REMOTE_USER and retrieve the Invoice_Number of this new record 17. For each of the upgrade options selected above where the option is not 0 or False, insert a new record in the Invoice_Item table with an appropriate description and the value of the option field multiplied by the corresponding value from the System Defaults table. 18. Calculate the total cost of the Invoice by taking the sum of all Invoice_Item records for this particular Invoice_Number and put this value in the Invoice_Amount field for the corresponding Invoice. 19. Display the Invoice information using the format supplied in the invoice.tpl file. Direct the user to print this invoice and reference it when sending in their cheque.
Possible Exit Conditions and Return Values	<ol style="list-style-type: none"> 1. If getLogin returns a NULL value, the user is redirected to the user/index.html page. 2. Otherwise, the user is supplied with an invoice of the total cost of their upgrades
Sign Off by:	Project Manager

vol/listapps.cgi

Procedure Model Type: ()Web Page <u>(X)CGI Script</u> ()Shared Library ()System API Name: vol/listapps.cgi Assigned to: Reference:	
Description	This script lists the applications for accounts currently in a pending state and asks a volunteer to approve or delete the application
Implementation Skills	PERL, CGI, SQL
Parameter List	
Called By:	vol/index.html vol/listapps.cgi
Can Call:	add_ppp() add_db(dbname) add_quota(mnt_point, Mb) add_ssl(virtual_domain_name) add_listserver(list_name) vol/index.html /index.html
Function Description	<p>If the volunteer has selected the Delete button from a previous invocation of the form:</p> <ol style="list-style-type: none"> 1. Get the User_Number from the hidden field, then delete the User table record for this User, any Member table record for this user, any Upgrade_Request table record for this user where the Upgrade_ApprovedBy field is NULL, any Invoice table record where the Invoice_Number was in the Upgrade_Request record just deleted, and any Invoice_Item where the Invoice_Number is the Invoice_Number of the Invoice table record just deleted. <p>If the volunteer has selected the Approve button from a previous invocation of the form, perform the following steps:</p> <ol style="list-style-type: none"> 2. Call the sys/add_account configuration command, passing it the username (User_UserName), password (User_InitialPassword), and if there is a Member table record with this User_Number, the Member_Type value for this User_Number (one of "Institution" or "Individual"). If there is no corresponding Member table record for this User_Number, then it will pass the value of "User" as the value of the Member_Type parameter. 3. If the Upgrade_toPPP is True, then call the add_ppp configuration command to add this user to the radiusd server. 4. If the Upgrade_toDB is True, then call the add_db command with the name from the Upgrade_DBName value and the Username of the user making the request passed as parameters. 5. If the Upgrade_Quota1 is greater than 0, then call the add_quota command with the parameters Sys_Quota1_Mnt, the value of the Upgrade_Quota1 multiplied by 1024, and the user's Username who is making the request. Repeat this step for Quota2 and Quota3. 6. If Upgrade_toVirtualDomain is True, then call add_virtualdomain with the name from the Upgrade_VirtualDomainName and the user's Username and Groupname passed as parameters. The Groupname should be set to "users", the default UNIX user group for new users. 7. If Upgrade_toSSL is True, then call add_ssl with the value of the Service_VirtualDomainName (if one exists) or the value of the Upgrade_VirtualDomainName field. 8. If Upgrade_toAliases is True, then verify that the Upgrade_AliasGPFNAddress is not a UserName assigned to another user by checking whether it exists in the User or Member tables and that it is not an existing Alias for another user by checking whether it exists in the E-mail Alias table. If the alias is not assigned, then call the add_alias command with the values from the

	<p>Upgrade_AliasExtAddress and Upgrade_AliasGPFNAddress as parameters.</p> <ol style="list-style-type: none"> 9. If Upgrade_toListserver is True, then call the add_listserver command with the name from the Upgrade_ListserverName and the user's Username as parameters. 10. Update the User table record, the Upgrade table record, the Member table record and the Services table record by inserting the value of the REMOTE_USER variable into the User_EnteredBy, Upgrade_ApprovedBy, Services_EnteredBy, and Member_EnteredBy fields. Set the User_IsActive field to "Active" from "Pending". Set the User_ExpiryDate to December 31 of the current calendar year. Set the User_Status_Msg to "User account approved: <timestamp>" where timestamp is the current date. 11. If the User is a member, then ensure that a record in the Services table exists for this user, even if that record contains only the default values for each field (i.e. the member has not upgraded to any additional services). 12. Create a new receipt table record with the value of the Invoice_Number matching this User_Number, the User_Number, the current timestamp as the Receipt_Date, and the values from the CGI form for the values of Receipt_Amount, and Receipt_Payment_Type and REMOTE_USER as the Receipt_Entered_By. <p>For all invocations of the script including the first:</p> <ol style="list-style-type: none"> 13. Select all records in the User table where the User_IsActive field = "Pending" 14. For each record found in step 1, list the following information in a formatted table for each record. Each table should contain an HTML form with a hidden field containing the User_Number. 15. Display the User_FirstName, User_LastName, User_Initial, User_Salutation, User_Organization (if exists), User_Title (if exists), User_StreetAddress1, User_StreetAddress2 (if exists), User_City, User_Province, User_Country, User_PostalCode, User_HomePhone (if exists), User_WorkPhone (if exists), and User_Age. If any of the required fields listed in the specifications for user/apply.html are left blank, a warning message should be displayed to the volunteer. 16. In addition to the above fields, if a record exists in the Member table with this same User_Number perform the following additional steps: <ul style="list-style-type: none"> • Check if there is a record in the Upgrade table for this User_Number where Upgrade_ApprovedBy is NULL. If there is, display a series of checkboxes and textfields listing the following fields from that table: Upgrade_toPPP (checkbox) Upgrade_toDB (checkbox) and show Upgrade_DBName (textfield) if Upgrade_toDB is True Upgrade_Quota1 (textfield) Upgrade_Quota2 (textfield) Upgrade_Quota3 (textfield) Upgrade_toVirtualDomain (checkbox) and show Upgrade_VirtualDomainName (textfield) if Upgrade_toVirtualDomain is True Upgrade_toSSL (checkbox) Upgrade_toAliases (textfield) and show Upgrade_AliasExtAddress (textfield) and Upgrade_AliasGPFNAddress(textfield if Upgrade_toAliases is an integer > 0 Upgrade_toListserver (checkbox) and Upgrade_ListserverName (textfield) if Upgrade_toListserver is True • Get the Invoice table record with this User_Number and display the Invoice_Date and Invoice_Amount. • Display a textfield named Receipt_Amount and a radio button array named Receipt_Payment_Type with values "Cash", "Cheque", "In-Kind", "Visa", or "MasterCard". 17. Display a "Approve" Submit Button and a "Delete" Submit Button for each Form.
<p>Possible Exit Conditions</p>	<ol style="list-style-type: none"> 1. If there are no pending applications, the list should be empty and

and Return Values	volunteers can click a link to return to vol/index.html or the GPFN home page. 2. Any application from the list can be left it is current pending state, deleted from the system, or approved.
Sign Off by:	Project Manager

vol/listupgrades.cgi

Procedure Model	
Type: () Web Page (<u>X</u>) CGI Script () Shared Library () System API	
Name: vol/listupgrades.cgi	
Assigned to:	
Reference:	
Description	This script lists the upgrades for existing accounts where an unapproved Upgrade_Request table record exists.
Implementation Skills	PERL, CGI, SQL
Parameter List	
Called By:	vol/index.html vol/listupgrades.cgi
Can Call:	lib/add_alias lib/change_username Prototype-User (non-login account profile) Prototype-Ind (non-login account profile) Prototype-Inst (non-login account profile) add_ppp() add_db(dbname) add_quota(mnt_point, Mb) add_ssl(virtual_domain_name) add_listserver(list_name) vol/index.html /index.html
Function Description	<p>If the volunteer has selected the Delete button from a previous invocation of the form:</p> <ol style="list-style-type: none"> Get the User_Number from the hidden field, then delete the corresponding Upgrade_Request record, and update the corresponding User table User_Status_Msg with the string "Upgrade request declined on <date> by <REMOTE_USER> because: <Delete_Reason>." where the appropriate values are inserted into the variables. Delete the corresponding record in the Invoice table and Invoice_Item table for this Upgrade_Request. <p>If the volunteer has selected the Approve button:</p> <ol style="list-style-type: none"> If Upgrade_User-Ind or Upgrade_User-Inst fields are True and no Member table record exists for this User_Number: insert a new record in the Member Table, setting the User_Number to User_Number, Member_Since to today's date, Member_EnteredBy to REMOTE_USER, Member_UserName to the value of UserName supplied from the form, and the Value of Member_Type to "Institution" in the case of a Upgrade_User-Inst request or "Individual" for a Upgrade_User-Ind request. Call the add_alias command with parameters User_UserName as the old address and Member_UserName as the new address to map to. Call the change_username command to rename the user's home directory and alter the appropriate entries in the /etc/passwd and /etc/shadow files. Call the edquota command with parameter "Prototype-Ind" or "Prototype-Inst" depending on the value of the Member_Type field. If there is a corresponding record in the Member table for this User_Name: <ul style="list-style-type: none"> If the Upgrade_toPPP is True, then call the add_ppp configuration command to add this user to the radiusd server.

	<ul style="list-style-type: none"> ▪ If the Upgrade_toDB is True, then call the add_db command with the name from the Upgrade_DBName value and the Username of the user making the request passed as parameters. ▪ If the Upgrade_Quota1 is greater than 0, then call the add_quota command with the parameters Sys_Quota1_Mnt, the value of the Upgrade_Quota1 multiplied by 1024, and the user's Username who is making the request. Repeat this step for Quota2 and Quota3. ▪ If Upgrade_toVirtualDomain is True, then call add_virtualdomain with the name from the Upgrade_VirtualDomainName and the user's Username and Groupname passed as parameters. The Groupname should be set to "users", the default UNIX user group for new users. ▪ If Upgrade_toSSL is True, then call add_ssl with the value of the Service_VirtualDomainName (if one exists) or the value of the Upgrade_VirtualDomainName field. ▪ If Upgrade_toAliases is True, then verify that the Upgrade_AliasGPFNAddress is not a UserName assigned to another user by checking whether it exists in the User or Member tables and that it is not an existing Alias for another user by checking whether it exists in the E-mail Alias table. If the alias is not assigned, then call the add_alias command with the values from the Upgrade_AliasExtAddress and Upgrade_AliasGPFNAddress as parameters. ▪ If Upgrade_toListserver is True, then call the add_listserver command with the name from the Upgrade_ListserverName and the user's Username as parameters. ▪ Update the Upgrade table record and the Services table record by inserting the value of the REMOTE_USER variable into the User_EnteredBy, Upgrade_ApprovedBy and Services_EnteredBy fields. ▪ Create a new receipt table record with the value of the Invoice_Number matching this User_Number, the User_Number, the current timestamp as the Receipt_Date, and the values from the CGI form for the values of Receipt_Amount, and Receipt_Payment_Type and REMOTE_USER as the Receipt_Entered_By <p>For all invocations of the script including the first:</p> <ol style="list-style-type: none"> 4. Select all records in the Upgrade_Request table where the matching record in the User table with the same User_Number does <i>not</i> have User_IsActive = "Pending" 5. For each record found in step 1, list the following information in a formatted table for each record. 6. Each table listed in step 2 should contain an HTML form with a hidden field containing the User_Number. 7. Display the User_FirstName, User_LastName, User_Initial, User_Salutation, User_Organization (if exists), User_Title (if exists), User_StreetAddress1, User_StreetAddress2 (if exists), User_City, User_Province, User_Country, User_PostalCode, User_HomePhone (if exists), and User_WorkPhone (if exists). 8. In addition to the above fields, display the following additional values: Upgrade_User-Ind or Upgrade_User-Inst (as a radio button) Upgrade_toPPP (checkbox) Upgrade_toDB (checkbox) and show Upgrade_DBName (textfield) if Upgrade_toDB is True Upgrade_Quota1 (textfield) Upgrade_Quota2 (textfield) Upgrade_Quota3 (textfield) Upgrade_toVirtualDomain (checkbox) and show Upgrade_VirtualDomainName (textfield) if Upgrade_toVirtualDomain is True Upgrade_toSSL (checkbox) Upgrade_toAliases (textfield) and show Upgrade_AliasExtAddress (textfield) and Upgrade_AliasGPFNAddress(textfield) if Upgrade_toAliases is an integer > 0
--	--

	<p>Upgrade_toListserver (checkbox) and Upgrade_ListserverName (textfield) if Upgrade_toListserver is True</p> <ol style="list-style-type: none"> 9. Get the Invoice table record with this User_Number and display the Invoice_Date and Invoice_Amount. 10. Display a textfield named Receipt_Amount and a radio button array named Receipt_Payment_Type with values "Cash", "Cheque", "In-Kind", "Visa", or "MasterCard". 11. Get the Invoice table record with this User_Number and display the Invoice_Date and Invoice_Amount. 12. Display a textfield named Receipt_Amount and a radio button array named Receipt_Payment_Type with values "Cash", "Cheque", "In-Kind", "Visa", or "MasterCard". 13. Display a textfield named "Delete_Reason". 14. Display a "Approve" Submit Button and a "Delete" Submit Button for each Form.
Possible Exit Conditions and Return Values	<ol style="list-style-type: none"> 1. If there are no pending applications, the list should be empty and volunteers can click a link to return to vol/index.html or the GPFN home page. 2. Any application from the list can be left it is current pending state, deleted from the system, or approved.
Sign Off by:	Project Manager

vol/geninvoice.cgi

<p>Procedure Model</p> <p>Type: ()Web Page <input checked="" type="checkbox"/>CGI Script ()Shared Library ()System API</p> <p>Name: vol/geninvoices.cgi</p> <p>Assigned to:</p> <p>Reference:</p>	
Description	This script taking parameters from the form on invoice.html generates invoices or statements for one or all system users
Implementation Skills	PERL, CGI, SQL
Parameter List	Invoice_Count User_UserName Type_Switch Medium_Switch
Called By:	vol/invoices.html vol/geninvoices.cgi
Can Call:	vol/index.html
Function Description	<p>Depending on the values of the Type_Switch and Invoice_Count parameters, create a temporary directory named after the current process_id in which to store the following generated invoices or statements. Only one of the following six steps will be used for each invocation of this script:</p> <ol style="list-style-type: none"> 1. If Type_Switch = 'NewInvoice' and Invoice_Count = 'all', first ensure that the Sys_Last_Annual_Cycle is no more than 16 months after the current date³. If this test is successful, then get a list of all records in the User table where User_IsActive = 'Active' and User_Expiry_Date is less than the Sys_Last_Annual_Cycle. For each of these User records, create a new Invoice record, inserting the User_Number as the foreign key, the current date as the Invoice_Date, the Invoiced_Medium as the value of the Medium_Switch field, and the Invoice_EnteredBy as the value of REMOTE_USER. <ul style="list-style-type: none"> ▪ If the User IsMember field = 0, create a new invoice for this user. This Invoice should have a single Invoice_Item with Item_Amount

³ We don't mind invoices going out in September for the coming year, but let's make sure that we don't have two volunteers thinking this way or we'll be billing people for two years down the road.

	<p>= Sys_Reg_User_Price and Item_Description = "Renewal of Free Internet Account".⁴</p> <ul style="list-style-type: none"> ▪ If the User_Is_Member field= 1, denoting that the user is a member, then determine if the User is an Individual or Institutional member by querying the Member table for this user and create a new Invoice record with an Invoice_Item with Item_Description of either "Institutional Membership Renewal" or "Individual Membership Renewal" and an Item_Amount of either "Sys_Ind_Mem_Price" or "Sys_Inst_Mem_Price" respectively. ▪ If the User_Is_Member field = 1 and the user has a corresponding record in the Services table, for each service listed in the Service Table, multiply any non-zero value by the respective Price in the System_Defaults table and create a new Invoice_Item record with this product in the Item_Amount field and the Service Description in the Item_Description field, and update the Invoice_Number and User_Number. ▪ Once the Membership and all Services have been invoices for a user, obtain the sum of all Invoice_Item entries for this Invoice_Number and put this value in the Invoice_Amount field ▪ Increment the sys_Last_Annual_Cycle by 1 year to be December 31 of the end of the current billing cycle. <p>2. If Type_Switch = 'NewInvoice' and Invoice_Count = '1', get the User_UserName value from the CGI script and query the User table to get the corresponding User_Number for this UserName. For this User_Number, create a new Invoice record, inserting the User_Number as the foreign key, the current date as the Invoice_Date, the Invoiced_Medium as the value of the Medium_Switch field, and the Invoice_EnteredBy as the value of REMOTE_USER.</p> <ul style="list-style-type: none"> ▪ If the User_IsMember field = 0, create a new invoice for this user. This Invoice should have a single Invoice_Item with Item_Amount = Sys_Reg_User_Price and Item_Description = "Renewal of Free Internet Account". ▪ If the User_Is_Member field= 1, denoting that the user is a member, then determine if the User is an Individual or Institutional member by querying the Member table for this user and create a new Invoice record with an Invoice_Item with Item_Description of either "Institutional Membership Renewal" or "Individual Membership Renewal" and an Item_Amount of either "Sys_Ind_Mem_Price" or "Sys_Inst_Mem_Price" respectively. ▪ For each service listed in the Service Table, multiply any non-zero value by the respective Price in the System_Defaults table and create a new Invoice_Item record with this product in the Item_Amount field and the Service Description in the Item_Description field, and update the Invoice_Number and User_Number. Once all Services have been invoices for the user, obtain the sum of all Invoice_Item entries for this Invoice_Number and put this value in the Invoice_Amount field. ▪ Once the Membership and all Services have been invoices for the user, obtain the sum of all Invoice_Item entries for this Invoice_Number and put this value in the Invoice_Amount field. <p>3. If Type_Switch = 'ExistingInvoice' and Invoice_Count = 'all', get a list of all invoices from the Invoice_Table where there is no corresponding Receipt table entry and where 'User_IsActive = 'Active'.</p> <p>4. If Type_Switch = 'ExistingInvoice' and Invoice_Count = '1', get the User_UserName value from the CGI script and query the User table to get the corresponding User_Number for this UserName. Get a list of all invoices from the Invoice_Table where there is no corresponding Receipt table entry and where the Invoice table User_Number is the current User_Number.</p> <p>5. If Type_Switch = 'Statement' and Invoice_Count = 'all', get a list of all Invoices and Receipts for each User where User_IsActive = 'Active'.</p>
--	--

⁴ This requirement is intended to allow the Free-Net to query existing registered users to determine if they still want their free e-mail and dial-up account. The invoice should have some information for the user to either print the invoice, sign it and send it in, or else e-mail a confirmation that they want to keep their account.

	<p>Format each statement using the stmt.tpl template file such that all invoice and receipt entries are combined and then sorted in an oldest-date-first list.</p> <p>6. If Type_Switch = 'Statement' and Invoice_Count = '1', get a list of all Invoices and Receipts for the User_Number corresponding to the User_UserName value. Format the statement using the stmt.tpl template file such that all invoice and receipt entries are combined and then sorted in a oldest-date-first list.</p> <p>Regardless of which of the above six steps was selected, get the value of the Medium_Switch parameter:</p> <ol style="list-style-type: none"> 1. If the Medium_Switch = 'E-mail' format each file in the temporary directory created above as an e-mail message and send the statement or invoice via e-mail to <User_UserName>@gpfn.ca. Delete each temporary file after it is sent. 2. If the Medium_Switch = 'Screen', format each file in the temporary directory created above as an HTML table, formatting it according to the invoice.tpl or statement.tpl file and then zipping it and then inserting a link in the returned page so the volunteer can download the file using a HTTP transport. Delete each temporary file after it is incorporated into the new HTML page. <p>Delete the temporary directory created above.</p>
Possible Exit Conditions and Return Values	The script should always return successfully
Sign Off by:	Membership Committee or board designate

vol/receipts.cgi

Procedure Model	
Type: ()Web Page (X) <u>CGI Script</u> ()Shared Library ()System API	
Name: vol/receipts.cgi	
Assigned to:	
Reference:	
Description	This script gets a list of all unpaid invoices and displays them to the volunteer so that the volunteer can apply receipts against them.
Implementation Skills	PERL, CGI, SQL
Parameter List	
Called By:	vol/index.html vol/receipts.cgi
Can Call:	vol/index.html
Function Description	<p>If the script has been called by a previous invocation of this same script as determined by the value of the Submit button being "Apply Receipt":</p> <ol style="list-style-type: none"> 1. Create a new Receipt record with the values of the Invoice_Number hidden field, the User_Number for this invoice, the Receipt_Date being the current date, the Receipt_Amount being the amount entered from the CGI form, the Receipt_Payment_Type being the value entered from the CGI form, and the Receipt_EnteredBy field being set to REMOTE_USER. 2. If the sum of a Receipt_Amounts in the Receipt table where the Invoice_Number corresponds to the Invoice_Number passed through the CGI form is equal to or greater than the value of the Invoice record's Invoice_Amount for this Invoice_Number, then set the User_Expiry_Date one year past its current date. (i.e. the Invoice has been paid in full). <p>For all invocations through this script, even the first, perform the following steps:</p> <ol style="list-style-type: none"> 3. Get a list of all Invoices from the Invoice table where the User_IsActive = 'Active' of the corresponding User_Number and where the sum of all Receipt_Amounts in the Receipt table with this Invoice_Number is less than the Invoice_Amount (i.e. the invoice has not been fully paid). 4. Display a table to the volunteer. For each Invoice returned in step 1, create a row in a table that contains a form. Set the Invoice_Number as a hidden field within each form. 5. Each row will also include: the User's first and last name, the User's username, and the value of the User_Since, User_Expiry_Date, and the value of the corresponding Member_Type field in the corresponding Member table. Also display the Invoice_Date, the Invoiced_Medium, the Invoice_Amount, and the sum of any Receipt_Amounts already applied against the Invoice. Present a textfield with the default value set to the difference of the Invoice_Amount less the sum of all Receipt_Amounts applied against this Invoice. Display a radio button array with the values "Cash", "Cheque", "In-Kind", "Visa", and "MasterCard". Each row will also include a submit button with the name "Apply Receipt"
Possible Exit Conditions and Return Values	The script should always return successfully. If no outstanding invoices exist, an empty list should be presented. Volunteers can always click on a link to take them to the vol/index.html or GPFN main page.
Sign Off by:	Membership Committee or board designate

Shared Library Procedures

lib/uname_test.pl

Procedure Model Type: ()Web Page ()CGI Script <u>(X)Shared Library</u> ()System API Name: lib/uname_test.pl Assigned to: Reference:	
Description	This script verifies that a username passed as a string to the function would be a good username. The test of goodness requires that the proposed username is unique to the system (i.e. no existing user has this name), and that the username is comprised of only alphabetic characters, numbers, underscores and dots (period), and that the username is between 4 and 16 characters long. For the purpose of the test, uppercase and lowercase characters are equivalent so that if the user has chosen a username of upper-case characters, these characters will be replaced with their lower-case equivalents.
Implementation Skills	PERL
Parameter List	String containing the proposed username
Called By:	user/validate.cgi user/upgrade.cgi lib/uname_generate.pl
Can Call:	
Function Description	<p>Verify the appropriateness and uniqueness of the proposed username by performing the following steps:</p> <ol style="list-style-type: none"> 1. Check that the username is between 4 and 16 characters in length and contains only characters, digits, underscores and dots (periods). If it fails this test, exit this function and return an appropriate error description. 2. Convert the proposed username to all lowercase letters. 3. Check that the username is unique within the system. If this test fails, exit the function and return an appropriate error description. This test of uniqueness is met if: <ul style="list-style-type: none"> ▪ There is no existing entry in the /etc/passwd file with the same username ▪ There is no existing entry in the /etc/mail/aliases file such that the proposed username was a previous system username that has since been remapped to a new username. ▪ There is no other entry in the User table or Member table with this username such that the same username is in a pending state awaiting another user's agreement or remittance. 4. Return a null length string if the proposed username is acceptable to the system.
Possible Exit Conditions and Return Values	<ol style="list-style-type: none"> 1. If the username is not appropriate, a descriptive error message is returned as a scalar variable. 2. If the username is appropriate, the function returns a null length string.
Sign Off by:	Project Manager.

lib/uname_generate.pl

Procedure Model	
Type: ()Web Page ()CGI Script <u>(X)Shared Library</u> ()System API	
Name: lib/uname_generate.pl	
Assigned to:	
Reference:	
Description	This function returns a string containing the next available sequential username following the format aannn where "a" denotes an alphabetic, lowercase character and "n" denotes a digit. For example, if the last username assigned by this function was aa999, the function will test that ab000 is not assigned and if it is free, will assign it. Otherwise, the function will test for ab001 and so on.
Implementation Skills	PERL
Parameter List	none
Called By:	user/validate.cgi
Can Call:	lib/uname_test.pl /usr/local/csuite/etc/lastacct
Function Description	<ol style="list-style-type: none"> 1. Open the /usr/local/csuite/etc/lastacct file if one exists. If this is the first time the script is ever called, this file will have to be created. 2. The contents of the lastacct file should include the last created username. If it does not, assume that the lastacct has value "aa000." Read this username into a variable and increment it according to the algorithm described in the description section above. Write this new incremented value back to the lastacct file, deleting all previous content. 3. Test this newly incremented value using the uname_test.pl function. If this new username passes this test return this new username to the calling procedure. Otherwise, repeat steps 2 and 3 until this test is successful.
Possible Exit Conditions and Return Values	The function returns the value of the next available sequentially assigned username as a scalar value.
Sign Off by:	Project Manager

lib/pw_check

Procedure Model Type: ()Web Page ()CGI Script <u>(X)Shared Library</u> ()System API Name: lib/pw_check Assigned to: Reference:	
Description	This function takes a string parameter and runs it against the system pw_check utility to see if it is a sufficiently strong password. If not, the function returns a string describing the weakness(es) of the password.
Implementation Skills	C
Parameter List	password- a string containing the password string to test
Called By:	user/validate.cgi
Can Call:	crack.h
Function Description	<pre> #include <stdio.h> #include <stdlib.h> #include <crack.h> int main(int argc, char *argv[]) { int i; char* pw_check; char* password; char* dict_path = "/usr/lib/cracklib_dict"; char* null_string = ""; char* bad_usage = "Usage: pw_check [password]"; if (argc != 2) { fputs(bad_usage, stdout); return (-1); } password = argv[1]; pw_check = FascistCheck(password, dict_path); if (pw_check != NULL) { fputs(pw_check, stdout); } else {fputs(null_string, stdout);} exit(0); } Usage within a PERL or Shell Script: \$output=`pw_check([password])`; if (\$output eq "") { # password is good } else { # password is bad. Reason is stored in \$output } </pre>

Possible Exit Conditions and Return Values	<ol style="list-style-type: none"><li data-bbox="690 199 1364 325">1. The program returns a -1 if an insufficient number of parameters were passed to the command. The description of the proper way to call the program should be passed through to the calling routine using the stdout handle. Check the contents of the \$output variable in the above function description example.<li data-bbox="690 325 1364 451">2. The program returns a 0 if the program is properly called. If the password is sufficiently strong, then an empty string is passed back through the stdout handle. Otherwise, a text string describing the problems with the password are listed. Since multiple problems can exist, these lines will be separated with a newline character ("\n").
Sign Off by:	Project Manager

lib/Login.pm

Procedure Model Type: ()Web Page ()CGI Script <u>(X)Shared Library</u> ()System API Name: lib/Login.pm Assigned to: Reference:	
Description	<p>This module manages login sessions for the system. Its public methods are called by CGI scripts to determine if the user is properly authenticated on the system and has currently valid credentials.</p> <p>The Constructor assumes that the user has been authenticated by the login.cgi script. As such, it accepts the call to create the session token, but does no additional checks on the authenticity of the user.</p>
Implementation Skills	PERL, SQL, MySQL
Parameter List	Uusername, access_level
Called By:	user/login.cgi user/upgrade.cgi user/myacct.cgi user/quota.cgi user/passwd.cgi user/forward.cgi user/history.cgi user/filter.cgi
Can Call:	Http_Sessions database.
Function Description	<p>HTTP_Sessions database Sessions table:</p> <pre> +-----+-----+-----+-----+-----+-----+ Field Type Null Key Default Extra +-----+-----+-----+-----+-----+-----+ User_Name varchar(20) PRI Session_Tok varchar(32) Access_Level int 0 TimeStamp timestamp(14) YES NULL +-----+-----+-----+-----+-----+-----+ </pre> <p>Constructor(UserName, Access_Level) method:</p> <ol style="list-style-type: none"> 1. Take the username and access_level from the parameter list passed to the function. 2. Open the sessions database and delete any previous entry for this same username. 3. Create a session token by MD5 base64 encoding the current timestamp. 4. Insert a new record into the Sessions table with this User_Name, Session_Token, and Access_Level. 5. Populate and return a Login object with the User_Name, Session_Token and Access_Level. This object is then used by the calling program to send a Login cookie back to the user's browser using the format: "<username>:<access_level>:<session_token>" <p>getLogin() method:</p> <ol style="list-style-type: none"> 1. Get a reference to the CGI query object and extract the Login cookie from this object. 2. If no cookie exists, return a null Login object. Otherwise, split the Login cookie using the colons into its username, access_level and session_token values. 3. Query the Sessions table for a matching token and if one exists populate a new Login object with the values from the database. Otherwise, return a null Login object. <p>getUserName(Login) method:</p> <ol style="list-style-type: none"> 1. Using the Login object passed as a parameter, return the Username string property of the object.

	<p>getAccessLevel(Login) method:</p> <ol style="list-style-type: none"> Using the Login object passed as a parameter, return the Access_Level integer property of the object. <p>destroy(Username) method:</p> <ol style="list-style-type: none"> Delete any record in the Sessions table with this Username and nullify any cookie stored on the browser by returning a new Login cookie with an empty string for its value.
Possible Exit Conditions and Return Values	<p>A Login object with undefined properties indicates that the system failed to find any credentials for this user. This test is usually performed by using the getLogin() method to get a Login object and then calling the getUsername(Login) method with the Login object returned by the getLogin method. If the getUsername method returns an undefined value, then the credentials do not exist or there is a system problem.</p> <p>Note that this class does not test for the age of a set of credentials although such a test can be included by having the getLogin method test the age of the session token by looking at the timestamp value in the Sessions table.</p>
Sign Off by:	Project Manager.

lib/auth_user

Procedure Model	
Type: () Web Page () CGI Script <u>(X) Shared Library</u> () System API	
Name: lib/auth_user	
Assigned to:	
Reference: http://www.gpfn.ca/~daryle/code/index.html	
Description	This suid program takes the username and password and returns a 0 if the username and password match the system password for the stated user. Any other return code indicates an error.
Implementation Skills	C, PAM
Parameter List	Username and password as string values
Called By:	user/login.cgi
Can Call:	PAM system libraries
Function Description	<pre> Example of a proper function call from a calling PERL script: output=`/usr/local/csuite/lib/auth_user \$user \$passwd ` /***** ** Library functions to interact with the Linux-PAM ** ** modules in order to update a user's password on ** ** the system. ** ** ** ** Make sure you add the following lines to the ** ** pam.conf file (or equivalent): ** ** cs_password auth required ** ** /lib/security/pam_unix_auth.so ** ** cs_password account required ** ** /lib/security/pam_unix_acct.so ** ** cs_password password required ** ** /lib/security/pam_unix_passwd.so ** ** cs_password session required ** ** /lib/security/pam_unix_acct.so ** ** ** ** Author: Daryle Niedermayer (dpn) ** ** daryle@gpfn.ca ** ** Date: 2002-06-17 ** ** ** *****/ #include <stdio.h> #include <stdlib.h> #include <security/pam_appl.h> #include <security/pam_misc.h> #define CS_BAD_DATA -2 #define CS_BAD_USAGE -1 #define CS_SUCCESS 0 #define COPY_STRING(s) (s) ? strdup(s) : NULL /* DEFINE STATIC EXTERNAL STRUCTURES AND VARIABLES SO THAT THEY ONLY HAVE SCOPE WITHIN THE METHODS AND FUNCTIONS OF THIS SOURCE FILE */ static char* service_name = "cs_password"; static char* user; static char* old_password; static char* new_password; static int PAM_conv (int, const struct pam_message**, struct pam_response**, void*); static struct pam_conv PAM_converse = {PAM_conv, NULL}; </pre>

```

/*****
** PAM Conversation function
**
*****/

static int PAM_conv (
    int num_msg,
    const struct pam_message **msg,
    struct pam_response **resp, void *appdata_ptr
) {
    int replies = 0;
    struct pam_response *reply = NULL;
    reply =
        malloc(sizeof(struct pam_response) * num_msg);
    if (!reply) return PAM_CONV_ERR;

    for (replies = 0; replies < num_msg; replies++) {
        if (! strcmp(msg[replies]->msg, "Password: "))
            reply[replies].resp =
                COPY_STRING(old_password);
        if (! strcmp(msg[replies]->msg,
            "(current) UNIX password: "))
            reply[replies].resp =
                COPY_STRING(old_password);
    }
    *resp = reply;
    return PAM_SUCCESS;
}

/*****
** MAIN PROCEDURE
**
*****/
int main(int argc, char *argv[]) {

    /* DEFINITIONS */
    pam_handle_t*    pamh = NULL;
    int              retval;
    char*            pw_check;
    char*            dict_path =
"/usr/lib/cracklib_dict";

    /* DETERMINE IF VARIABLE COUNT IS CORRECT */
    if (argc != 3) {
        printf("Usage: auth_user <USER> <PASSWORD>\n");
        exit (CS_BAD_USAGE);
    }

    /* PARSE PARAMETERS FROM INPUTS */
    user          = argv[1];
    old_password = argv[2];

    if (!(user && old_password && strlen(user) &&
        strlen(old_password)))
        exit (CS_BAD_DATA);

    /* GET A HANDLE TO A PAM INSTANCE */
    retval = pam_start(service_name, user,
        &PAM_converse, &pamh);

    /* IS THE USER REALLY A USER? */
    if (retval == PAM_SUCCESS)
        retval = pam_authenticate(pamh, 0);
    else return retval;

    /* IS USER PERMITTED ACCESS? */
    if (retval == PAM_SUCCESS)
        retval = pam_acct_mgmt(pamh, 0);
    else return retval;

    /* CLEAN UP OUR HANDLES AND VARIABLES */
    if (pam_end(pamh, retval) != PAM_SUCCESS)

```

	<pre> pamh = NULL; else return retval; exit (CS_SUCCESS); } </pre>
Possible Exit Conditions and Return Values	<p>This function returns a number of possible values:</p> <ol style="list-style-type: none"> 1. CS_BAD_DATA = -2 is returned if any of the username or passwords are null length strings or do not exist. 2. CS_BAD_USAGE = -1 is returned if an invalid number of parameters is passed to the function. 3. CS_SUCCESS = 0 is returned if the function successfully authenticated the user with the supplied parameters. 4. Any other positive integer indicates a PAM error. The meaning of the error value can be determined by referencing the /usr/include/security/_pam_types.h header file.
Sign Off by:	Project Manager

System Configuration Procedures

sys/add_account

Procedure Model Type: ()Web Page ()CGI Script ()Shared Library <u>(X)System API</u> Name: sys/add_account Assigned to: Reference:	
Description	This system configuration function runs with suid root permissions. It adds a new account to the system with the value of the username field passed in as a parameter, then sets the initial password on this account to the value of the password field passed in as a parameter. Finally, it uses the system edquota command to set the user quota on this new account to the value stored in the corresponding prototype account.
Implementation Skills	PERL, C
Parameter List	Username, password, and member_type.
Called By:	vol/listapps.cgi
Can Call:	/usr/bin/passwd /usr/sbin/useradd /usr/sbin/edquota -p prototype-user (non-login account profile) prototype-ind (non-login account profile) prototype-inst (non-login account profile)
Function Description	<ol style="list-style-type: none"> 1. The script will be executed using suid root permissions. 2. It will call the system useradd command passing it the username value contained in its parameters. 3. It will call the system passwd command with parameters username and password. 4. Call the system edquota -p command passing it the username value and, depending on the value of the Member_Type command, the "prototype-ind", "prototype-inst", or "prototype-user" account names.
Possible Exit Conditions and Return Values	<ol style="list-style-type: none"> 1. If the user or system is unsuccessful in executing its parts, it will return a -1 if it is unable to invoke the useradd command, -2 if it is unable to set the password, and -3 if it is unable to set the disk quotas. 2. If the alias is successfully created or replaced, the function returns a 0.
Sign Off by:	Project Manager

sys/add_alias

Procedure Model Type: ()Web Page ()CGI Script ()Shared Library <u>(X)System API</u> Name: sys/add_alias Assigned to: Reference:	
Description	This system configuration function reads the system /etc/mail/aliases file, checks that no current entry matching old_address exists and if one does exist, deletes it. It then adds a new entry mapping old_address to new_address and rebuilds the newaliases database. The function must NOT

	be allowed to alter any of the system aliases. Because editing the aliases file and rebuilding new alias databases are protected procedures, this function will need suid permissions.
Implementation Skills	PERL, C
Parameter List	old_address and new_address where old_address is the address to which mail is sent and new_address is the address to which mail is delivered.
Called By:	vol/listapps.cgi vol/listupgrades.cgi
Can Call:	/usr/bin/newaliases /etc/aliases log/csuite
Function Description	<ol style="list-style-type: none"> 1. Open the /etc/aliases file for reading and check whether any existing mapping from old_address exists. If this mapping exists, verify whether it exists above the lines marked: ##### ## Lines below this line are managed by the ## ## /usr/local/csuite/sys/add_alias command. ## ## Do NOT edit aliases below this line. ## 2. If the mapping exists above this section, return a -1 error. The mapping is a system mapping and cannot be altered by this script. 3. If the mapping exists below the commented section, delete the entry from the /etc/aliases file. Enter the details of this deletion in the csuite log file. 4. Add a new entry to the /etc/aliases file, inserting the entry alphabetically after the above comment section. The new entry should have the form: <old_address>: <tab><new_address> 5. Enter the details of this insertion in the csuite log file. 6. Run the /usr/bin/newaliases command to recompile a new aliases database.
Possible Exit Conditions and Return Values	<ol style="list-style-type: none"> 3. If the user or system is attempting to create or replace a system alias, the function returns a -1. 4. If the alias is successfully created or replaced, the function returns a 0.
Sign Off by:	Project Manager

sys/change_user

Procedure Model	
Type: ()Web Page ()CGI Script ()Shared Library (X)System API	
Name: sys/change_user	
Assigned to:	
Reference:	
Description	This function renames the user's home directory from old_username to new_username, effecting this change in the system password and shadow files and adding a system alias mapping the old_username to new_username. Because of the permissions involved, this command must be set to suid root.
Implementation Skills	C
Parameter List	old_username, new_username
Called By:	vol/listupgrades.cgi
Can Call:	sys/add_alias /usr/sbin/usermod log/csuite
Function Description	<ol style="list-style-type: none"> 1. Use the system's usermod command to move the user's home directory to the new location and move the login name to the new login name. 2. Enter an entry in the log/csuite directory noting these changes. 3. Call the add_alias command to add a new alias mapping from the old_username to the new_username. 4. Return 0.
Possible Exit Conditions and Return Values	This function should always return 0.
Sign Off by:	Project Manager

sys/add_virtualdomain

Procedure Model	
Type: ()Web Page ()CGI Script ()Shared Library (X)System API	
Name: sys/add_virtualdomain	
Assigned to:	
Reference:	
Description	This function is a stub to add a new virtualdomain to the webserver configuration. It currently manages a manual workflow process but can be automated in the future.
Implementation Skills	PERL
Parameter List	Domainname, Username, Groupname
Called By:	vol/listapps.cgi vol/listupgrades.cgi
Can Call:	log/csuite
Function Description	<ol style="list-style-type: none"> 1. E-mail the request to the webserver administrator to set up a new virtual domain. The root directory of the new virtualdomain and all child directories should be owned by the user given in the Username parameter and the group given in the Groupname parameter.
Possible Exit Conditions and Return Values	Currently, this function will always be successful (returning 0). In the future, it will return a -1 if the virtual domain name is already taken.
Sign Off by:	Project Manager

sys/add_ppp

Procedure Model	
Type: ()Web Page ()CGI Script ()Shared Library <u>(X)System API</u>	
Name: sys/add_ppp	
Assigned to:	
Reference:	
Description	This function is a stub to grant ppp access to a new or existing member. It currently manages a manual workflow process but can be automated in the future.
Implementation Skills	PERL
Parameter List	username
Called By:	vol/listapps.cgi vol/listupgrades.cgi
Can Call:	log/csuite
Function Description	<ol style="list-style-type: none"> 1. E-mail the request to the system administrator to set up ppp access for a user. 2. Log this entry to the log/csuite file
Possible Exit Conditions and Return Values	Currently, this function will always be successful (returning 0). In the future, it will return a -1 if ppp access is not permitted for some reason.
Sign Off by:	Project Manager

sys/add_db

Procedure Model	
Type: ()Web Page ()CGI Script ()Shared Library <u>(X)System API</u>	
Name: sys/add_db	
Assigned to:	
Reference:	
Description	This function is a stub to create a new database and provide read/write access to a new or existing member. It currently manages a manual workflow process but can be automated in the future.
Implementation Skills	PERL
Parameter List	Database_name, Username, Password
Called By:	vol/listapps.cgi vol/listupgrades.cgi
Can Call:	log/csuite
Function Description	<ol style="list-style-type: none"> 1. E-mail the request to the system administrator to set up a new database with access for the user with the Username parameter. This user should have complete access with grant privileges to the database with the password supplied through the Password parameter. 2. Log this entry to the log/csuite file
Possible Exit Conditions and Return Values	Currently, this function will always be successful (returning 0). In the future, it will return a -1 if the database creation failed for some reason.
Sign Off by:	Project Manager

sys/add_quota

Procedure Model	
Type: ()Web Page ()CGI Script ()Shared Library (X)System API	
Name: sys/add_quota	
Assigned to:	
Reference:	
Description	This function is a stub to add disk quota to a user. It currently manages a manual workflow process but can be automated in the future.
Implementation Skills	PERL, C
Parameter List	Mount_point, Mb to increase, Username
Called By:	vol/listapps.cgi vol/listupgrades.cgi
Can Call:	log/csuite /usr/sbin/setquota
Function Description	<ol style="list-style-type: none"> 1. E-mail the request to the system administrator to add an additional number of Mb of file quota on the specified mount point for a user given in the parameter Username. 2. Log this entry to the log/csuite file
Possible Exit Conditions and Return Values	Currently, this function will always be successful (returning 0). In the future, it will return a -1 if the setquota command fails for some reason.
Sign Off by:	Project Manager

sys/add_ssl

Procedure Model	
Type: ()Web Page ()CGI Script ()Shared Library (X)System API	
Name: sys/add_ssl	
Assigned to:	
Reference:	
Description	This function is a stub to add ssl access to a virtual domain. It currently manages a manual workflow process but can be automated in the future.
Implementation Skills	PERL, C
Parameter List	Virtual_domain
Called By:	vol/listapps.cgi vol/listupgrades.cgi
Can Call:	log/csuite
Function Description	<ol style="list-style-type: none"> 1. E-mail the request to the system administrator to add an SSL enabled directory to the virtual domain specified in the parameters. 2. Log this entry to the log/csuite file
Possible Exit Conditions and Return Values	Currently, this function will always be successful (returning 0). In the future, it will return a -1 if the configuration is not permitted for some reason.
Sign Off by:	Project Manager

sys/add_listserver

Procedure Model	
Type: <input type="checkbox"/> Web Page <input type="checkbox"/> CGI Script <input type="checkbox"/> Shared Library <input checked="" type="checkbox"/> System API	
Name: sys/add_listserver	
Assigned to:	
Reference:	
Description	This function is a stub to add a new mailing list to the listserver. It currently manages a manual workflow process but can be automated in the future.
Implementation Skills	PERL, C
Parameter List	Listname, Username
Called By:	vol/listapps.cgi vol/listupgrades.cgi
Can Call:	log/csuite
Function Description	<ol style="list-style-type: none"> 1. E-mail the request to the listserver administrator to add a new mailing list to the listserver with the specified name. 2. Log this entry to the log/csuite file. Username should be added to the logging entry for audit and tracing purposes.
Possible Exit Conditions and Return Values	Currently, this function will always be successful (returning 0). In the future, it will return a -1 if the configuration is not permitted for some reason.
Sign Off by:	Project Manager