Network Services Laboratory Project
Student Originated Software Undergraduate Program
The Evergreen State College
1999 – 2000 Project Notebook

Staff:

Erik Ordway	**Lead Programmer, Team Lead**
Brook Hatch	**Content Manager, Web Dev**
Randy Jacobson	**Chief Designer, Curriculum Dev**
Patrick Butler-Monterde	**Content Developer**

NSL (Network System Laboratory), a one year project of Student Originated Software (SOS) at Evergreen State College, is a continuation of the network lab previously built and operated by a group of 1998-1999 students as a group contract.

We inherited a heterogeneous collection of hardware, and we built two Windows NT servers, two OpenBSD firewalls, a Linux workstation, a Windows 95 workstation, and a Linux based web server . We were able to make all of these machines communicate over Ethernet using TCP/IP protocol.

Given the context of our academic program, we decided to make an application utilizing our resources. Our application is based on HTML written by Randy, Brook, and Patrick and utilizes a Java control system written by Erik.

The Presentation Manager is a system designed to allow a developer to write content modules about any subject and have features like Keyword and Glossary searches.

The system is written in Java, JavaScript and HTML and makes use of the Sun Microsystems Java 1.2.2 Plug-in. It has been developed and tested using Apache web server and Microsoft Windows clients using at least Netscape 4 and MSIE 4.

The Presentation Manager applet and associated HTML, files can be stored and accessed both locally and from a network fileshare with full functionality. This makes the product available offline and could even be distributed on one floppy disk (core and two modules).

The system is not currently compatible with Mac and Linux clients but could be re-engineered to be so.

Table of Contents

Project Team Members

Erik Ordway – Lead Programmer, Team Lead

Learning objectives
A very general understanding of NT administration.
How VPN's are implemented, in particular IPSec network tunneling.
How to route across multiple networks.
Database access from applications

Role:
My role in the project has been two fold. I have acted as the project manager and as the lead programmer. In my role as the project manager I have focused on building the lab infrastructure and it's integration with the campus network. In my role as the lead programmer I have focused on implementing and sometimes changing the design that the project has developed.

Responsibilities:
In my role as the project manager I have been working with the school and outside sources to gather new equipment for the lab. I have also been working with Evergreen Computing Services to add and improve network services to the lab. In my role as the lead developer I am responsible for making sure that the design for the Presentation Manager and the content modules is a workable one.

Accomplishments:
I have helped acquire for the lab five used workstation class computers and four Intel PCs. Some of these machines are being added to the present network and some will be for the following years projects to add. In the area of networking services I have been able to provide network connectivity to the lab, and a number of stable machines to provide service to the other machines. These services include a VisualAge Java development environment, a web server for sharing personal work and log entries, and a firewalled connection to the campus network and the outside world. In the role of Lead Programmer I have implemented, in Java, the first stage of the Presentation Manager system and designed the format for many files used by it.

Contributions:
A lot of time.

Brook Hatch – Content Manager

Learning Objectives
Web data base integration
Java
User Interface Design
UNIX hosting: telnet, ftp, nntp
VPN

Roles
Content Manager and Program Secretary. Responsible for presentations, reports, logs. Lead HTML mentor. Project evangelist. Responsible for software fair booth.

Responsibilities
Responsible for presentations, reports, logs. Leadership role in the development of suitable material for the Presentation Manager. Web design, graphics, desktop publishing, NT Domain Administration. Primary mentor for all Microsoft products used in Netlab.

Accomplishments
Built two machines from parts. Set up Domain Name Service, DHCP, NT domain user login and file sharing. Worked with Erik in DNS interoperability with SAMBA on Linux. In a liaison capacity with the College Network administrator, facilitated necessary firewall and network switching changes for Netlab.

Contributions
Participated in the design of the Presentation Manager and produced the BIOS submodule for Module One. Participated in the inception discussions with Neal Nelson, Torleiv Flatebo-Ringer, and Dave Metzler and the NSL team. Assisted with the conceptualization of the proper subnet masking scheme to use in the lab.

Randy Jacobson –Chief Designer, Curriculum Dev

Learning Objectives
Randy would like to get out of this project a working knowledge of computers: the components of a PC, network functionality, how to set and change the access rights, non-Windows operating systems, printer problems, the differences between network systems, virtual networking, tunneling, and cross-platform communications.

Role:
Randy's main role in the NSL team is in development for future iterations of this lab. He developed the outline of both workshop #1 and workshop #2, as well as coming up with the basic design for the workshops themselves. He researched what an educationally sound module should contain. He presided over the development of workshop 1, and wrote workshop 2 by himself. Randy also came up with a draft of workshops 3 - 6 for next year's iteration of the net-lab to develop, as well as some ideas for workshops beyond that. Randy took a lead in soliciting outside help in content development by seeking the input of Thad Curtz and John Cushing into of what a module should consist.

Very early in the winter quarter he proposed the order of chapters in the *TCP/IP Illustrated* text that was used. He continually pushed this schedule, keeping on track.

Randy's final major role was in documentation. In the inception and early in the elaboration phase, he recorded the original design in Rational Rose. In the actual programming, Randy worked with Erik to produce a template for new modules. In doing so, he implemented the feature of having a glossary word definition accessed from within a module. Randy wrote the design specs for the program and the specs for creating a new module, soliciting outside people to read and review for readability, clarity, and completeness. He wrote the test plan and implemented the test. He wrote the directions for using the presentation manager, and he reviewed the final documentation.

Accomplishments:
Randy completed *TCP/IP Illustrated,* summarizing each chapter and completing at least half of the questions at the end of every chapter. With some initial input from Brook, he taught himself the basics of HTML and JavaScript. Erik taught him how to maneuver within a UNIX system, and provided him with many good references (book-wise and online). Erik also walked him through the installation of Linux and gave him a lesson in constructing an Ethernet cable from two RJ-45 plugs and a CAT-5 cable.

Additional Contributions:
Randy was the leader in figuring out the mathematical portions of the *TCP/IP Illustrated* exercises. The chief instance of this is in the net mask that is currently in use in the lab. Randy figured out how sub-net masks work, how to logically set up an intuitively nice sub-net, and what a good range for our lab sub-net should be.

He also took the lead in the presentations given to C of C and SOS. He developed, scripted, produced the visual aids, and was the main speaker in the network tutorial portion of the demonstration.

Patrick Butler Monterde – Content Developer

Learning Objectives
System Administration of UNIX OS: +Solaris + Linux + FreeBSD.
Setup and maintain a secure network: +Setup and configuration of Firewalls and proxy servers. +
Test different security programs, and utilities.
Compare security issues on different operative systems.
Learn about information warfare.
Setup a private network through the Internet (VPNs). Setup a Oracle8i database server and create a web intensive database system. Learn more about Database warehousing. Test some networking Java programs (using the JDBC and Java networking classes)

Role:
Create and write content for the modules
Network administration – Sun Solaris machines

Responsibilities
Create and Write the content for the modules
Take care of the Sun Microsystems workstations

Accomplishments
Linux administration and networking
Solaris administration
Installing OpenBSD, Solaris 7.0 & Linux (Red Hat & SuSe)
Create content (text and media) for module I
How to set up a firewall & router

Team Covenant

Over the course of Winter and Spring quarters we will:

- work through the networking assignments from the previous NetLab
  - We focused instead on developing curriculum more relevant to software engineering
- add new NetLab documentation
  - All documentation exists on our web page *http://grace.evergreen.edu/nsl*
- build a networking tutorial software product
  - Core engine and two modules developed
- produce documentation for the product
  - All documentation exists on our web page
- maintain a web page for the project
  - see URL above
- strive to make an equitable division of labor amongst the team members
  - This was a success.

Software Requirements Specification

Statement of Scope and Objectives

The NSL (Network System Laboratory), a one year project of Student Originated Software (SOS) at Evergreen State College, is a continuation of the network lab previously built and operated by a group of 1998-1999 students. Our goal is to provide an SOS project that builds on last years work and extends into the future. We will produce documentation to support next years iteration of this project, a simulated Wide Area Network, a Local Area Network, and an interactive tutorial system.

Due to the explosion of the Internet, there is high demand in today's business environment for networked systems. Building a network can be a daunting task. There are a multitude of Network Operating Systems to choose from and much confusion and hype surrounding the implementation of many current offerings. We feel that we can introduce into the market, a much needed interactive tutorial on how to set up a network. This system is capable of benefiting business users and virtually anyone else who would like to link two or more computers. We, like many other computer science aficionados, envision a world where all machines communicate; we can facilitate that by offering this content.

We do not intend to implement every possible network solution. Instead we will focus on Windows NT server and Linux. We have chosen these two because of their popularity, high profile and accessibility to us in our student environment. Our immediate customer is Neal Nelson, a faculty member of The Evergreen State College. We expect that the product we make will be modified by students in future iterations of the NetLab as well as being something that could potentially be shared with other educational institutions.

Narrative Overview of Problem Statement and System Requirements.

Must run on a stand alone machine

Platform interoperable - Win32, UNIX, MacIntosh

Comprehensive - covers all the steps

Constraints, Assumptions & Dependencies.

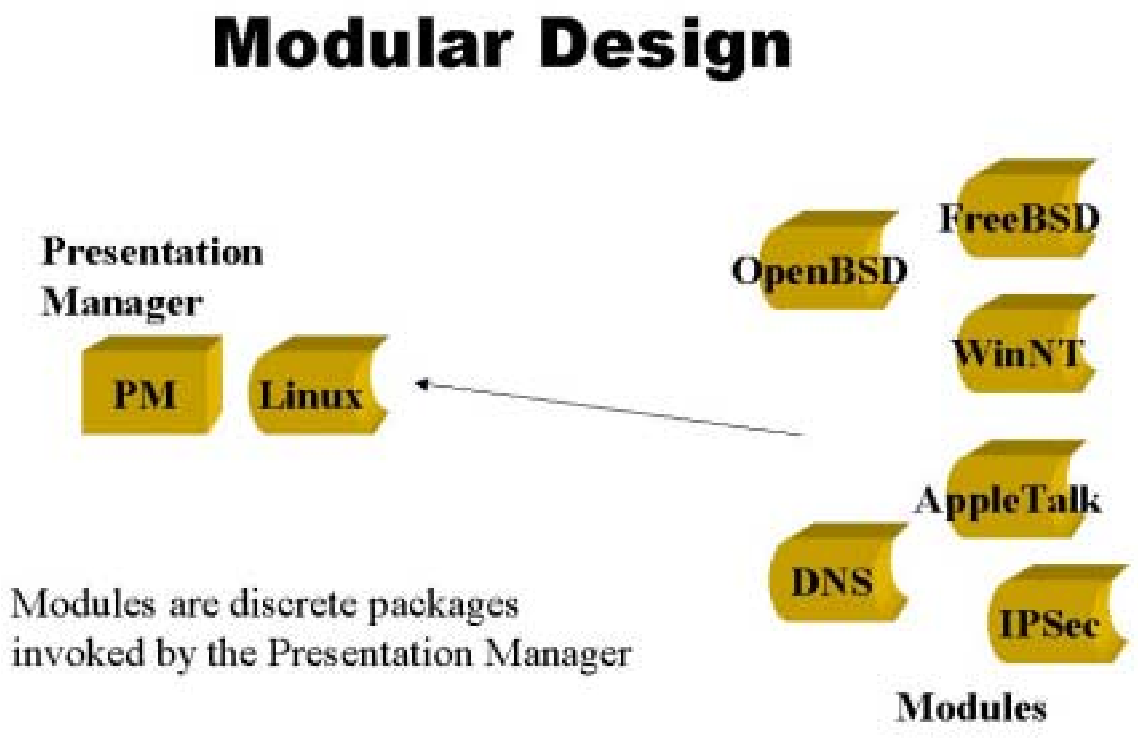
Written in Java and pure HTML (Hypertext Markup Language) for maximum browser compatibility

Utilize Rational Rose and the UML for design

Build a web page documenting/ advertising product to the world

Content should be valuable to a wide array of users.

Use NSL resources to develop content.



This design allows for easy extensibility. Developers can build content modules and drop them into the directory structure provided. A simple edit of two text files allows the applet to display and link to the new content.

Background Information on the Domain.

Bibliography.

TCP/IP Illustrated Volume 1 - The Protocols -W. Richard Stevens - Addison Wesley 1994 - ISBN 0-201-63346-9

Chapters 1-3 of Computer Networks Third Edition -Andrew S. Tanenbaum - Prentice Hall PTR - ISBN 0-13-349945-6

UNIX documentation available on the Web

Various RFCs (Request for Comments)

Previous NetLab assignments

Third party consultation as needed

Including Neal Nelson, Toleiv Flatebo-Ringer, Steve Dublin, and Dave Metzler.

Glossary

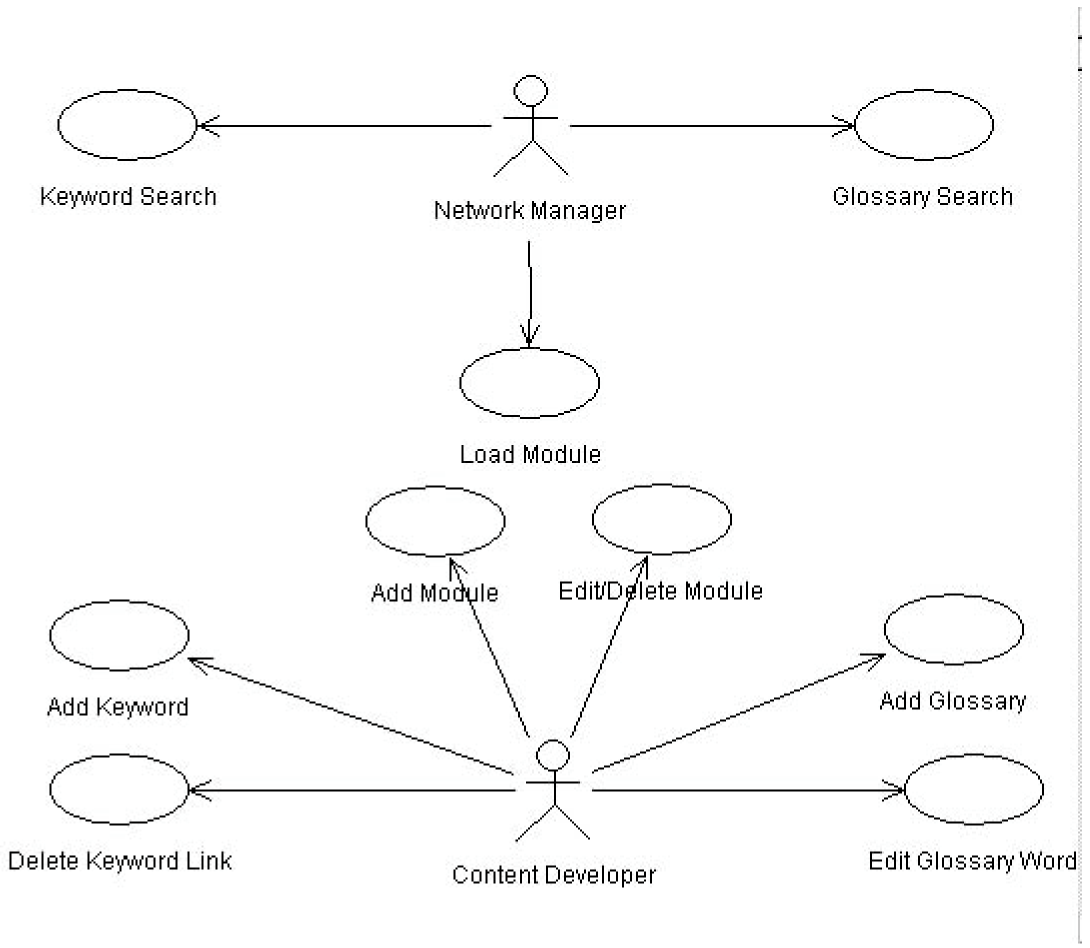
Module	A tutorial built to the specification of the Presentation Manager. It should be self-contained (no absolute references to itself) and have glossary, keyword, and about files.
Presentation Manager	A browser based applet responsible for organizing loaded modules. Within these modules will be additional information that the PM has the ability to display.
Module Listing	An applet that lists of properly loaded modules with links to the index page of the module and a short text describing what the module is about.
Nettie the Octopus	An image that has a link to the main applet. Clicking on Nettie will raise the glossary/keyword applet.
About file	A short HTML file that gives information on a module formatted to be used by the PM
Glossary file	An HTML file that has terms with definitions formatted to be used by the PM.
Keyword file	An HTML file that has hyperlinks attached to important terms from a module. It is formatted to be used by the PM.
Index.htm file	Unless otherwise directed, the first file looked for by a browser or by the PM when entering a directory.

User Characteristics.

Class name:
Network Manager
Stereotype: Actor
Documentation:
The primary enduser. This person will have a fairly sophistocated knowledge of how to use a computer. basic skills include knowing how to use a browser and how to load and save files.

Class name:
Content Developer
Stereotype: Actor
Documentation:
The person who designs and implements new modules into the Presentation Manager (PM). This person will need to know how to program web pages and how files are structured.

User scenarios for the major functions of the system as originally designed



Use Case name:
Keyword Search
Documentation:
A user enters in a keyword and is given a list of pages or modules to connect to that keyword.

1) The PM collects all keywords and all of the links and saves them in a file.
2) The user selects a keyword in the InfoApplet.
3) The user selects a link.
4) The PM goes into the appropriate module and pulls up the linked page.

Use Case name:
Load Module
Documentation:
The user chooses a module to use.
1) The PM lists all possible modules.

2) The user selects a module to load.
3) The PM goes into the chosen module and loads the initial page of that module.
4) The user navigates through the module at his or her own discression.

Use Case name:
Glossary Search
Documentation:
The user wishes to access the glossary for some purpose - 2 ways.
1) The user selects a word from within a module.
2) The PM shows the information linked to that word from within that module's glossary.

OR

1) The PM at start-up collects all of the glossarys into one place.
2) The user from the InfoApplet asks to view the glossary.
3) The PM shows all glossary entries in alphabetical order - regardless of module.

Use Case name:
Delete Keyword Link
Documentation:
The CD decides to no longer provide a link for a keyword.
1) The CD chooses a particular link to delete.
2) The PM provides specifications for deleting that link.

Use Case name:
Add Keyword
Documentation:
A CD chooses to link a keyword to a specific page, or
to a module.
1) The CD chooses a word to and a page to link.
2) The PM provides specifications for adding this link.

Use Case name:
Add Module
Documentation:
A CD has a developed module that he or she wishes to add to the possible modules in use.
1) The CD develops a module that can stand on its own to be used in a browser.
2) The PM provides the CD specifications for loading that module into the realm of the usable modules.

Use Case name:
Edit/Delete Module
Documentation:

The CD decides to edit a page, delete a page, or delete a whole module including the glossary.
1) The CD chooses a page to edit or delete.
2) The PM provides specifications for changing or deleting that page.

OR

1) The CD chooses a module to delete.
2) The PM provides specifications for deleting all traces of that module from the system.

Use Case name:
Add Glossary
Documentation:
A CD will add a glossary to a module that does not have a glossary.
1) The CD develops the content of a glossary for a module and decides where to link the glossary words to in the module.
2) The PM provides specifications as to creating a glossary file.

Use Case name:
Edit Glossary Word
Documentation:
The CD decides to change a glossary words link or verbage, or to add a new glossary word.
1) The CD develops the content of the change to happen.
2) The PM provides the specification as to how to do the necessary changes.

Feasibility Study.

Risk Assessment

Week 7 -	if design prototype doesn’t come together, redesign simpler
Week 11 -	if code prototype doesn’t work switch to HTML
Week 16 -	if Beta 1 is not complete, strip all optional components and/or go for a working skeleton/specification

Development Plan and project log

We have divided Winter and Spring quarter into these four segments of 5 weeks each. During the first segment, we will update the documentation for the lab that is our legacy, work through the course material provided by Neal, and do the first build of our network.

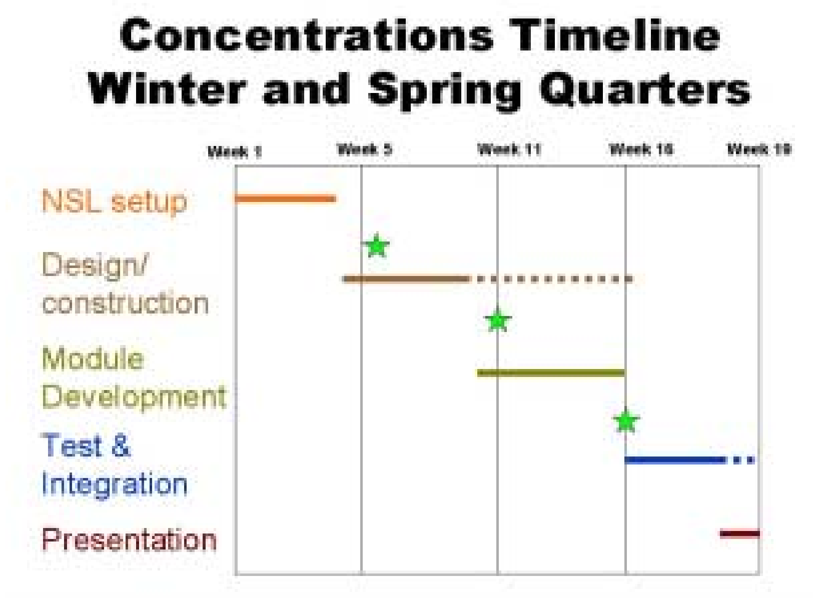
As the first 5 weeks comes to a close will have completed the inception phase.

In the next 5 weeks, we will design the core engine of the tutorial system and a default module using Rational Rose and the UML.

The third segment will be devoted to constructing our software using Java.

The final segment will be be mostly testing and integration. We will use the last two weeks of Spring quarter to prepare the product for demonstration and deployment.

Each of the 4 segments will contain elements of each of the 4 steps. We envision an iterative process. We will attempt to produce a working prototype by the end of Winter quarter. We chose Java for it's cross platform characteristics because we envision the final product running on any platform. There are many network operating systems and frequent upgrades and changes made to them. The preliminary architecture of the product emphasizes its extensibility and ease of maintenance.



Milestones

Week 7

Prototype Specification complete

Presentation Manager and Default Module

Week 11

Code Prototype Complete

Week 16

Beta 1 release,

Software Design Specification with design decisions, as appropriate.

Project Log

Week 1:	Presentation Manager Design, TCP/IP academics, network infrastructure planning.
Week 2:	PM Design, TCP/IP academics, network infrastructure planning.
Week 3:	PM Design, TCP/IP academics, network infrastructure implementation. Draft design specification created.
Week 4:	PM Design, TCP/IP academics, network infrastructure implementation.
Week 5:	PM Design, TCP/IP academics, web browser, Java and HTML research. Design specification complete.
Week 6:	PM Design, TCP/IP academics, work on Project Notebook. Decision to use a frameset in HTML pages.
Week 7:	Rapid prototype in JavaScript fails due to lack of major functionality of the language. Prototype module design inception.
Week 8:	Browsers Java not fully functional either. File hierarchy set.
Week 9:	Mover from AWT to Swing. Data Structures complete.
Week 10:	Erik really likes Swing. LearnPC module content design.
Week 11:	Features missing in standard (older) version of Swing in IBM Visual Age Java. LearnPC module creation.
Week 12:	Installed newer Swing in VAJ. LearnPC module creation.
Week 13:	Re-factored functionality out of GUI and moved to intermediary class to allow new GUI implementation. Module level glossary feature implemented.
Week 14:	Integration testing Java/HTML – bugs in code and modules. Splash designed and implemented.
Week 15:	Concepts of Computing presentation. Core functionality complete. System now working.
Week 16:	SOS presentation. Code maintenance and content support. Software fair infrastructure planning.
Week 17:	Software fair infrastructure implementation. TCP/IP module content design. Design specifications rough draft completed.
Week 18:	Software fair infrastructure implementation. TCP/IP module content creation. Test plan developed and implemented.
Week 19:	Modifications made to keyword presentation. NSL future planning. Documentation finalized. Curriculum developed.
Week 20:	Inventory, hardware disassembly, packing and storing.

Design Specs for the Network Lab Presentation Manager

Overview

The Presentation Manager (hereafter referred to as the “PM”) is a Netscape 4.X+ or IE 4.X+ accessible program on a Windows 95, 98, or NT platform with the Java 1.2.2 plug-in. To properly view the PM, the display settings must be greater than 800x600. The PM's stated purpose is to organize tutorials on some subject. These tutorials, or modules as they will be referred to in this document, are HTML pages linked together as web pages.

The PM has the following functionality:

- It dynamically displays a listing of loaded modules, each with a short text describing what it covers. This list provides a link to the opening page of these modules.
- It loads and alphabetizes all glossary words from each module to be accessed at any time. Glossary words have a link to the first page of the module from which they come and a definition for the word.
- It loads and alphabetizes all keywords from each module to be accessed at any time. Each keyword has a link to the specific page where the word is embellished.

These functions are always available by clicking on "Nettie the Octopus."

In addition, each module has links to its own glossary that will open the glossary to the selected word in a new window.

This specification has three major parts:

1) An overview of the program showing the **Directory/File Tree** and a brief explanation of each part.
2) A description of the **PM Loading & Presentation** – the portion of the spec that deals with the overall presentation of the PM and the back-end procedures within the PM. Included in this section are details of the program structure of the main applet, instructions on how to change the style of the keywords, the overall glossary, and the text about each of the modules, and directions on how to add a module or to change the order of the modules listed.
3) Instructions on the necessities of creating new **Modules**. This should show how to seamlessly add a new module that will implement all of the functions of the PM.

The Directory/File Tree

PM Loading & Presentation.

Individual Modules.

```
/Presentation_Manager           - directory that holds the whole program.
        index.htm               - loads the frameset.
        code.jar                - the code for all of the PM's applets.  See below.
        Splash.htm              - the main menu frame.
        top_frame.htm           - the title frame.
        corner_frame.htm        - the corner frame.  This frame houses a permanent link to the main menu and keeps alive
                                  the applet with the full glossary and keyword.
        aboutpm.htm             - a simple explanation on how to use the PM.  Links to the aboutPM module.
        /aboutPM                - a default module teaching about the PM.  Needs to be implemented.
                /content                - folder to hold content pages about the PM.  Needs to be developed.
        /modules                - holds all of the individual modules and files that format things for the PM.
                index.htm                 - a redirect back to the PM – should not be used.  See documentation for
                                            index.htm in Modules.
                modules.txt             - a text listing of the modules.  See below.
                about.htm               - the formatting for the about files.  See below.
                glossary.htm            - the formatting for the glossary files.  See below.
                keywords.htm            - the formatting for the keyword files.  See below.
                /ModuleName             - whatever name you wish to have for your module.
                        index.htm               - a redirect back to the PM – should not be used.  See below.
                        about.htm               - a short description of the module. See below.
                        glossary.htm            - the listing of glossary words for the module. See below.
                        keywords.htm            - the listing of the keywords and their links.  See below.
                        /content                - holds all of the actual content of a module.
                                index.htm               - the page the PM will load to start the module.  See below.
                                pageX.htm               - all of the individual pages named whatever you wish.  See
                                                          below.
                                top_frame.htm           - the title bar for the module.
                                moduleName.css          - the cascading style sheet (if used) for the module.
                                /images                 - images and media folders are for formatting
                                        image.X           purposes only.  It is good form to
                                /media                    separate them from the normal content.
                                        media.X
```

Regular = Directory   *Italic* = file name

PM Loading & Presentation

The PM concerns itself with the organization and listing of all loaded modules. It gathers and allows access to information from and about all of the modules, and has very little to do with the presentation of the individual modules, except providing the frameset from which to work.

The portions covered by this section of the specification should only be altered if you wish to adjust the appearance or performance of the module listing, the aggregate glossary, or the keyword portions of the program. To create or change a module, skip forward to **Modules**.

The PM will go through the following steps upon start up. The main applet will start, getting the modules.txt file for a list of all available modules. It will then use the directory names (the list from the modules.txt file) to load the *glossary*, *keywords*, and *about* files of each module. It will also load the format file for presenting each. The individual module’s *glossary* and *keywords* files will be parsed (taken apart), taking out the necessary information and eventually sorting it into alphabetical order. The main applet stays in memory in the form of Nettie the Octopus, who lives in the corner frame of the application, for use at any time. To access this applet, you must click on Nettie and a Java window will appear.

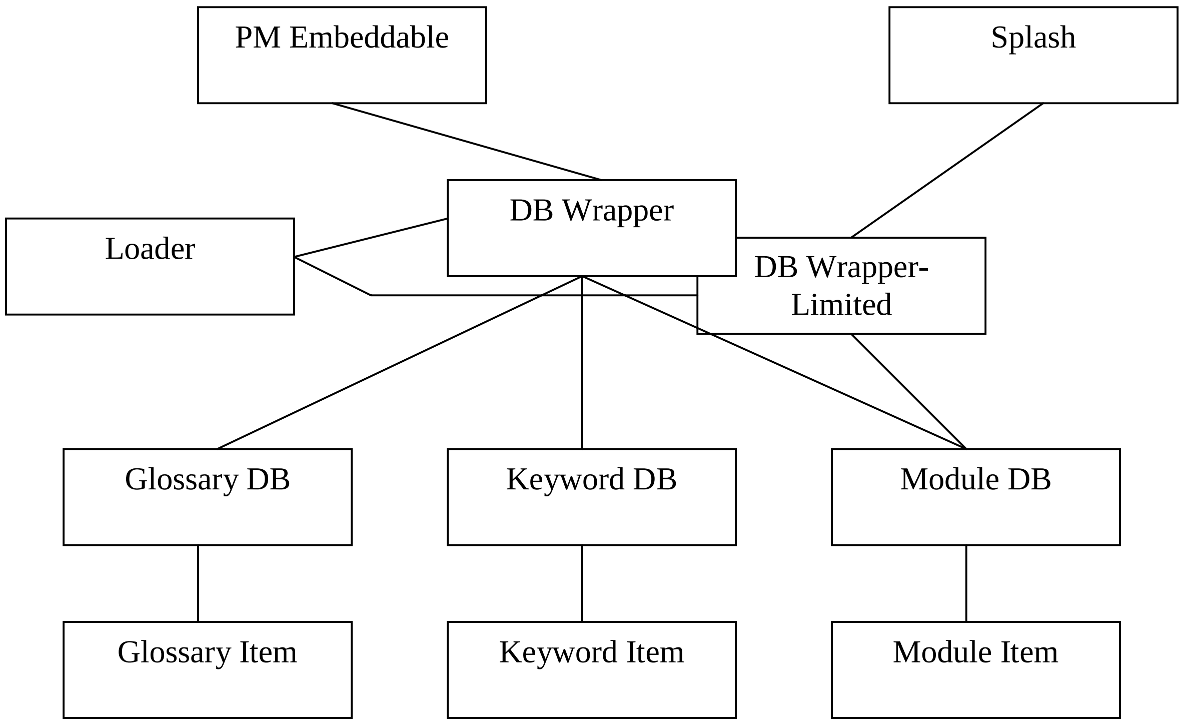
The module listing applet will load using the same methods but only retrieving the *about* file (not the *glossary* or *keywords* files). It will then display the modules in the order listed in the modules.txt file with each module's *about* text next to the module name/link.

The PM has five portions that will be covered in this specification:

1) The code.jar file. Included with this description will be a brief outline of the objects in the code and some of the more important functions.
2) The modules.txt file.
3) The about.htm file in the *modules* directory.
4) The glossary.htm file in the *modules* directory.
5) The keywords.htm file in the *modules* directory.

code.jar spec

Within the code.jar file is two applets that are very similar. The main applet is also called "PM Embeddable" as that is the name of the main object. The object map is below.



PM Embeddable

PM Embeddable invokes DB Wrapper, and hands off to it the location from which it was loaded, and the context of itself (what host it is on and the path on that host to the *Presentation_Manager* directory).

Splash

Splash is similar to PM Embeddable except for it invokes DB Wrapper-Limited instead of DB Wrapper. The purpose of Splash is to load the main menu when first loading the module.

DB Wrapper

DB Wrapper holds all of the information for an instance of running the applet together. It invokes Glossary DB, Keyword DB, and Module DB. It finds the format files for each of these objects and hands them off to the appropriate location. It then invokes Loader and hands this information off to it with the location and context. Loader will hand back the filled and sorted versions of all of the DB objects. DB Wrapper will kill Loader once it has done its job.

DB Wrapper - Limited

Limited is similar to DB Wrapper in function, but only authorizes Loader to get information for the Module DB object. The other 2 objects are kept void.

Generic DB

Generic DB holds the utility functions common to all of the DB classes, the search function, and the sorting algorithm used in the Keyword DB and the Glossary DB. The search algorithm goes in two steps. First it looks for an exact match. If no matches are found it will then take the first 3 letters of the word entered and find all words with the same first three letters. The major comparison function used in this algorithm is also used in the sorting algorithm, so be careful when changing it.

The sorting algorithm uses a divide and merge strategy. It will take a list, divide the list in half, sort the first half (using the same algorithm), sort the second half (using the same algorithm), and then merging the two sorted halves. Eventually by dividing the list of glossary words or list of keywords in half, you get a list of one and the merging puts them in alphabetical order.

Glossary DB

Glossary DB holds the contents of the glossary.htm file from the *modules* directory. It also holds pointers to all of the Glossary Items. They are not sorted until they are accessed for the first time. There is a Boolean field that indicates whether or not they have been sorted.

Keyword DB

Keyword DB holds the contents of the keywords.htm file from the *modules* directory. It also holds all of the Keyword Items. They are not sorted until they are accessed for the first time. There is a Boolean field that indicates whether or not they have been sorted.

Module DB

Module DB holds the contents of the about.htm file from the *modules* directory. It also holds all of the Module Items in the order they appear in the modules.txt file.

Generic Item

This class holds the common fields for all items, namely the module location.

Glossary Item

There are several Glossary Items, one for every glossary definition. Each glossary term may have more than one item if it is used in more than one module. Each item has a term, the module it came from, and the definition of the term.

Keyword Item

There are several Keyword Items, one for every keyword entry in each module. Each keyword may be listed more than once if it has a link to more than one page. Each item has a term, the module it came from, and the specific page (and anchor if appropriate) to which it is linked. It also has a flag that tells whether it is a normal or module level keyword.

Module Item

There are several Module Items, one for each module listed in the modules.txt file in the *modules* directory. Each item holds the name of the module and the module's about text.

Loader

Loader does most of the work that the applets need done upon loading. Within Loader are the parsers for the different DB objects.

The parser for the glossary words works by going through each glossary.htm file from within each module. It searches through the start of every line looking for the first <entry> tag. Upon finding the <entry> tag it will search for the <term> tag on the next line. It will then record everything after <term> and before </term> as the term. The parser will then disregard the rest of that line and start recording everything from the start of the next line to the end of the line before the </entry> tag as the definition. Finally the parser will look for the next <entry> tag and start all over again.

The parser for the keywords must differentiate between 2 types of keywords: module level keywords and normal keywords. Module level keywords have a <modkey> tag at the start of the line and a </modkey> at the end of the line. The *keywords* file should only have within it keywords, no header or formatting information, so the parser will start reading at the first line. If it sees the <modkey> tag it will store the term that follows as a module level term. If not, it will store the term as a normal keyword. Next, the parser looks for a <a href=…..> tag. The parser will take this as the page reference to be stored along with the keyword. The final thing that the parser will look for is the <word> tag. This tag tells the parser to record everything after that tag to the </word> tag as the keyword term. including spaces. The parser will then look at the beginning of the next line for the next term.

Both of these parsers are programmed to look for comments. Therefor if a line begins with "//", the parser will ignore everything until the next "//".

The parser for the *about* file looks for the two <!--about--> tags and records everything in between them as the about text. Everything else will be ignored.

modules.txt spec

modules/modules.txt holds the names of the modules' folders. They must have exactly the same spelling as the folder, otherwise the PM will not access the module properly. There also must be a blank line after the last entry – this is so that the listing of the modules with their *about* files on the main menu will be properly formatted.

Example:

```
--------
Module1
Module2
Module3

--------
```

Notice the blank line after the last module. Whenever a new module is created and integrated, this file needs to be updated.

The order in module.txt is the order in which the modules will be presented in the PM. If you wish to reorder the modules, you may do so in modifying this file.

modules/about.htm spec

The *about* file is a short description of what the module is about. It is used to give a user more information on what a module will cover.

There are two categories of *about* files. The first is in the *modules* directory and is common for all modules. This file contains the header and body information for displaying the information about each of the individual modules. This is done so that the PM has the same formatting for all of the about files. The PM will take the information from the second type of *about* files and place it where the "<!--break-->" tag is.

Example:

```
---------
<HTML>
<HEAD>
<TITLE>The Installed Modules</TITLE>
</HEAD>

<BODY LINK="#0000ff">
<B><FONT FACE="Arial" SIZE=4>
<P>The Installed Modules</P>
</FONT></B>
<FONT FACE="Arial" SIZE=4>
<!--break-->

</FONT>
</BODY>
</HTML>
---------
```

A couple of notes about this file.

1) The comment "<!--break-->" is very important. This is where the applet will put the text from the various modules. It must be on its own line for the parser.
2) The rest of the file is open for revision. You may format the output of these files in any way you wish. You may use cascading style sheets, different tags, different colors. The possibilities are endless and the choice is yours.

modules/glossary.htm spec

The *glossary* file is used in two separate applications, and therefore must be formatted for use in both. The PM combines the glossaries from all of the loaded modules for a user to access at any time. When accessed for the first time, they are organized into alphabetical order. It also provides a link to the index file of the module from whence the word came. In this way a user may go to the module and get more information.

The first glossary.htm file, within the *modules* directory, is very similar to the about.htm file in the same directory. It provides formatting for the PM's glossary page, with a <!--break--> in the middle for inserting the requested words. The <!--break--> must be on its own line.

Example:

```
---------
<HTML>
<HEAD>
<TITLE>The Glossary Items you were looking for.</TITLE>
</HEAD>

<BODY LINK="#0000ff">
<B><FONT FACE="Arial" SIZE=4>
<P>The Glossary Items you were looking for.</P>
</FONT>
</B><FONT FACE="Arial" SIZE=4>
<!--break-->

</Font>
</BODY>
</HTML>
---------
```

modules/keywords.htm spec

Keywords are implemented as simple hyperlinks to a specified location within a module. There are two defined keyword types. The first is a keyword that refers to the module as a whole and links to the *index* file of the module. The second is a reference to a specific page within a module.

Much like glossary and about, keyword has two files. The keywords.htm file within the *module* directory is a wrapper, placing a header and formatting information around the requested keywords. As previously noted, the <!--break--> must be on its own line.

Example:

```
---------
<HTML>
<HEAD>
<TITLE>The Keywords you requested.</TITLE>
</HEAD>

<BODY LINK="#0000ff">
<B><FONT FACE="Arial" SIZE=4>
<P> The Keywords you requested.</P>
</FONT>
</B><FONT FACE="Arial" SIZE=4>
<!--break-->

</Font>
</BODY>
</HTML>
---------
```

Modules

Each individual module is customizable to be formatted in any way that you see fit. There are only a few things that must be implemented for the PM to work with it. There must be a *content* folder with an index.htm file inside of it. This will be the first file that is called from the PM, but a redirect can be sent from this file to any other. The organization, feel, and content of the module are up to the writer. The other requirement is to keep the same frameset as is set in the PM - this ensures that "Nettie", the applet holding the consolidated glossary and keyword lists, does not die and does not need to be reloaded.

To use the full functionality of the PM there should be three other files created: the glossary.htm, about.htm, and keywords.htm files.

To create a new module, you only need to follow these specifications and add the name of the folder in which the module is held (the name of the module) to the modules.txt file. Each module should be completely self contained; there should be no absolute references; the module should be able to be transported from one location to another without loss of functionality.

index.htm and pageX.htm specs

Within the *content* directory of a module there must be an index.htm file. The PM uses this page when to load the module. Once the *index* file has been loaded, you may link to any page of any name in any order you wish.

The initial frameset has three frames: the corner frame of Nettie, the top frame that is to hold a modules title, and the main frame where content is to appear. To change the title frame upon loading a module, you may use the following JavaScript in the body tag of your *index* file.

```
<body onLoad="parent.frames[1].location='top_frame.htm';">
```

Upon loading a file, this script will load top_frame.htm into the second frame of the frameset (parent.frames[X] starts with 0 as the first frame loaded, 1 as the second, etc.). Then within the *content* directory, just format a file top_frame.htm to be the title of the module.

Within the *modules* directory and within each specific module's directory it is recommended to have an index.htm file. This file is to protect the data within those directories. No one should actually get to those files, but if something unexpected occurs, a safeguard should be implemented to protect the files in those directories. The readied solution is to redirect the browser back to the PM. The code for this redirect is as follows.

```
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta http-equiv=refresh content="0;URL=../../index.htm">
</head>
<body bgcolor="#FFFFFF">
</body>
</html>
```

The important line of code is the one that says **refresh content="0;URL=../../index.htm"** (which is changed to **../index.htm** in the *modules* directory). This tells the browser to go up two directories (or up one directory) and find the index.htm file. In this way if a person somehow accesses the wrong directory, he or she will end up back in the PM.

The full header of the normal pages within a module (the pageX.htm files) may be similar to this:

```
<head>
<title>Title of Page</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<!-- This javaScript function allows calls to the glossary and will ensure the glossary will be on top. Use with this
call: <a href="" onClick="goToWord('../glossary.htm#[anchor name]');return false;">
[glossary word]</a> -->
<script language="JavaScript">
<!--
function goToWord(glossaryWord) {
  var newWind = window.open(glossaryWord, "Glossary", "height=100,width=500,scrollbars=yes,resizable=yes");
 newWind.focus(); }
//-->
</script>
</head>
```

This includes the JavaScript function to call the module level version of the glossary.

about.htm spec

Within the main folder of each module is an *about* file that provides a short description of what the module covers. This file should have very little formatting and does not need headers. It does need a starting and ending <!--about--> tag around the portion of the *about* file that is to be displayed.

Example:

```
---------
<!--about-->
<P>
This module will take you through the basics of assembling a PC from scratch and installing an operating system.
</P>
<!--about-->
---------
```

This is all that is necessary, but you may add more if you wish. Just be very minimalistic about the formatting between the “<!--about-->” tags. Also, it is very important to have the tags on separate lines from the rest of the file, as this is how the parser will look for it.

glossary.htm spec

Within the main folder for each module should be a *glossary* file that provides the definition of important words within the module. The first use of this file, as described in the PM portion of this specification, has the PM stripping out the words and definitions, and then storing them for use in the main applet. The second use of the *glossary* file is within a module itself. A link is made within the pages of the module to open a second window with the glossary word in sight. This requires having an anchor for each word in the glossary. It also requires that the glossary.htm file within the module be nicely formatted.

Example:

```
---------
<HTML>
<HEAD>
<TITLE>Glossary Workshop #1</TITLE>
</HEAD>

<BODY LINK="#0000ff">
<B><FONT FACE="Arial" SIZE=4>
<P>Network Services Laboratory Glossary</P>
</font></B>
<DIR> <FONT SIZE=2>

<entry>
<a name="bsd"></a><term>BSD</term>
<P>(Berkeley Systems Design) The base set of code that has developed into FreeBSD,
    OpenBSD, NetBSD and BSDI. These are all UNIX based Operation Systems.</P>
</entry>
<entry>
<a name="linux"></a><term>Linux</term>
<P>A UNIX-like operating system. The OS's kernel was and is developed by Linus
    Torvalds and Alan Cox along with a large cast of global developers. In addition,
    the rest of the system and user level applications are freely available under
    GNU or GNU-like licenses. The OS runs on a large number of platforms. </P>
</entry>
<entry>
<a name="solaris"></a><term>Solaris</term>
<P>Sun Microsystems UNIX operating system for Sparc and i386 platforms.</P>
</entry>
</font> </dir>

</BODY>
</HTML>
---------
```

Notes on creating the glossary in order of the file.

1) This file is a fully fleshed out HTML file with header, body, and formatting.
2) Each <entry> tag must be on the start of its own line. The parser for the PM will not see the tag unless it is on the start of a line.
3) On the line below the <entry> tag, each entry needs to have an anchor attached to it(an <a name="…"> tag). This is so when the module accesses the *glossary* file, it can scroll down to the proper place to display the word.

4) The <term> and </term> must be on the same line as the anchor. The parser will look for the term there.
5) The definition must start on the line under the anchor. The parser, after stripping out the term will not look at anything else on that line. It will start at the beginning of the next line and start printing out the definition.
6) The </entry> tag must be on its own line. The parser will cease collecting a definition only when it sees this tag at the beginning of a line.

For the glossary words to be properly accessed within a module, a simple JavaScript function needs to be put in the header of every module page that points to a glossary word. A function that works is:

```
<script language="JavaScript">
<!--
function goToWord(glossaryWord) {
  var newWind = window.open(glossaryWord, "Glossary", "height=100,width=500,scrollbars=yes,resizable=yes");
 newWind.focus();
}
//-->
</script>
```

This script takes a [filename]#[anchor name] as an argument, opens a new window that is the given size and has scrollbars and can be resized, and puts this window on top of the main browser window (gives the new window the focus).

To then call this function within the body of a web page, use the following template.

> After you power on your computer, you should see a text message displaying
> your **<a href="" onClick="goToWord('../glossary.htm#bios');return false;">BIOS</a>**.

This function sets a hyperlink on the word BIOS to ../glossary.htm#bios. The ‘href = “”’ and the ‘return false’ statements are so that clicking on the link does not change the position on the page. The ‘../glossary.htm’ portion is needed because the *glossary* file is located one directory above the content page. The ‘#bios’ is referring to an anchor specified within the *glossary* file of ‘bios’.

keywords.htm spec

Within the main folder of each module should be a *keywords* file that provides links to pages. The keyword should represent an important concept that is illustrated in great detail on the page to which it is linked. There are two required tags within each line. The first is the <a href="…"> tag. This defines the word as a hyperlink and tells the browser where to go. This may be a page reference, or may be reference to an anchor within a page.

Each reference is of the style *content/pageX.htm*. The reason for having the reference include the content directory was for testing purposes. To test the links, pull up on a browser keywords.htm and click on all of the links. Any link that does not correctly pull up the appropriate page is not coded correctly.

The second required tag within each line is <word>…</word>. This allows the PM to know what the keyword term is, including spaces.

The keywords.htm file should not have a header or even be characterized as an <HTML> file. The parser will start reading from the first line. It also needs one blank line at the end of the file

Example:

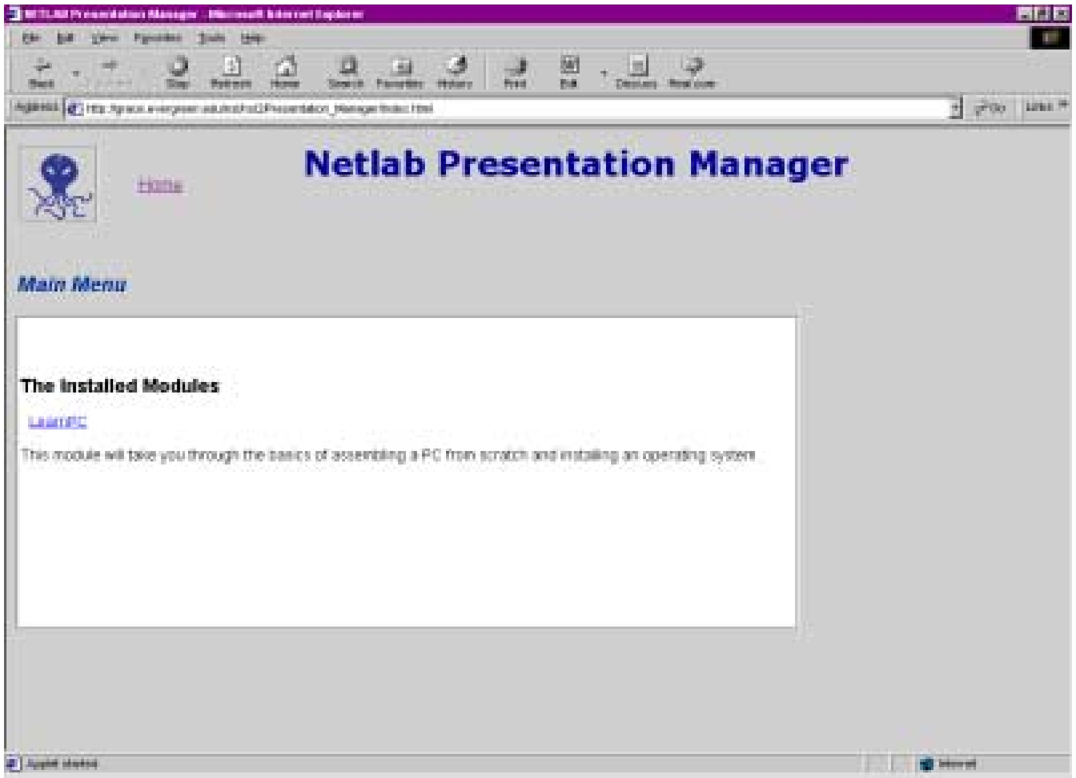
```
---------
<modkey><p><a href="content/index.htm"><word>PC Assembly</word></a></p></modkey>
<modkey><p><a href="content/index.htm"><word>Operating System</word></a></p></modkey>
<p><a href="content/body5.htm"><word>Operating System</word></a></p>
<p><a href="content/body1CPU.htm"><word>CPU (definition)</word></a></p>
<p><a href="content/body2b.htm"><word>CPU (assembly)</word></a></p>
<p><a href="content/body1Cac.htm"><word>Cache</word></a></p>
<p><a href="content/body1.htm#motherboard"><word>Motherboard (definition)</word></a></p>
<p><a href="content/body2b.htm"><word>Motherboard (assembly)</word></a></p>

---------
```

Note that "Motherboard" has an anchor attached to the link; this is permitted.

Navigable - standard GUI layout/buttons

Base HTML GUI



Home

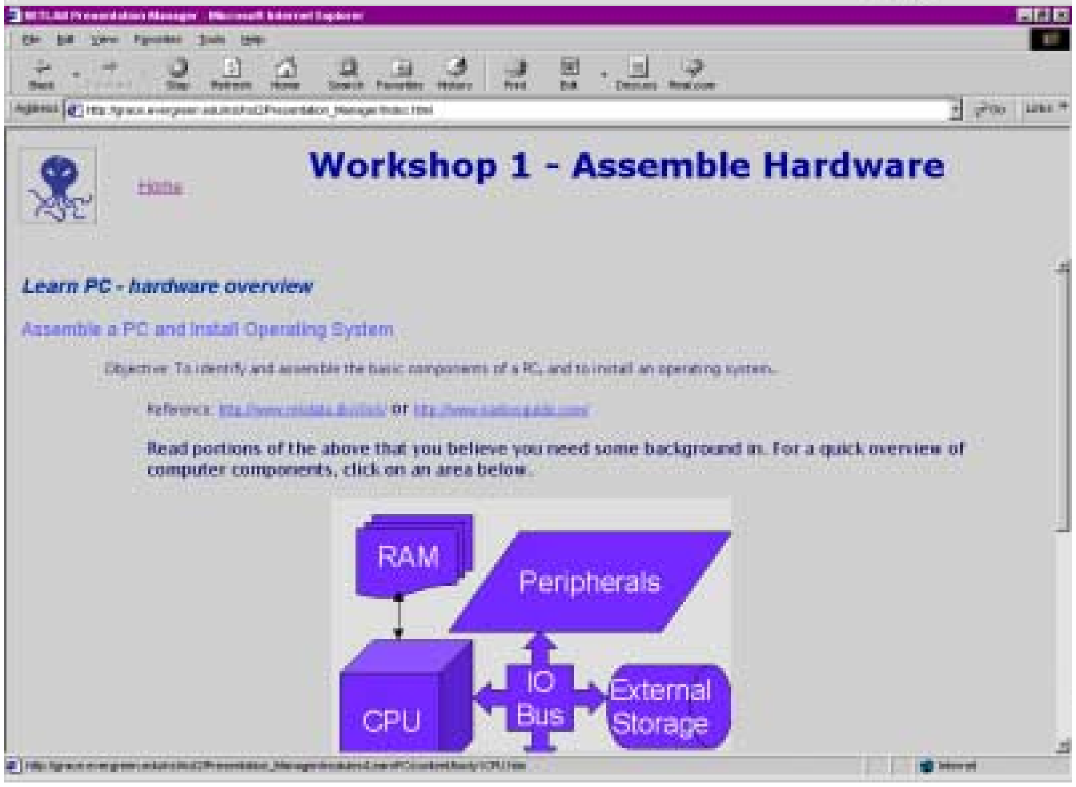
Netlab Presentation Manager

Main Menu

The Installed Modules

LearnPC

This module will take you through the basics of assembling a PC from scratch and installing an operating system.



Home

Workshop 1 - Assemble Hardware

Learn PC - hardware overview

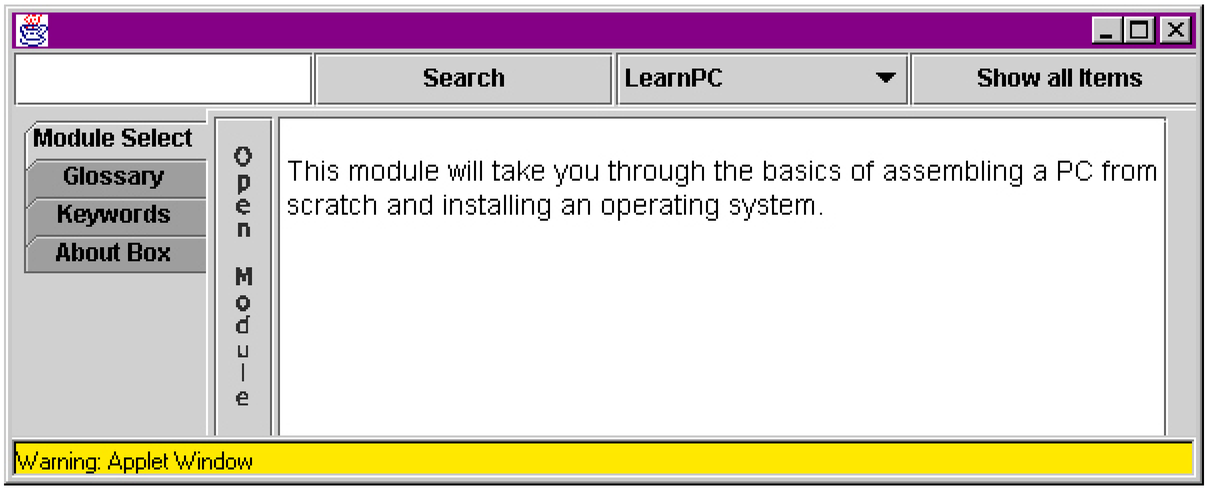
Assemble a PC and Install Operating System

Objective: To identify and assemble the basic components of a PC, and to install an operating system.

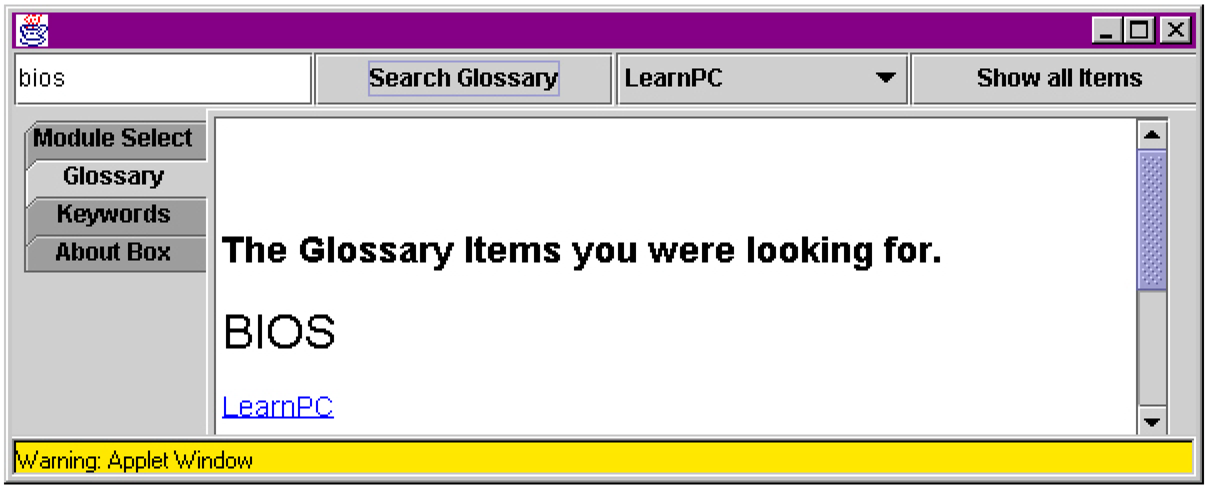
Reference: [illegible] or [illegible]

Read portions of the above that you believe you need some background in. For a quick overview of computer components, click on an area below.

GUI for Presentation Manager Applet



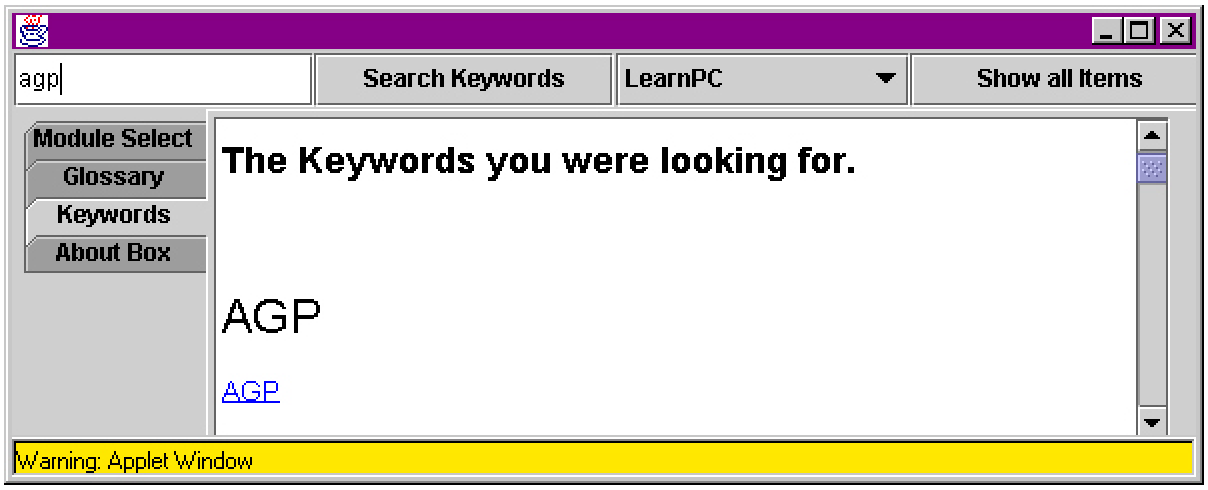
This module will take you through the basics of assembling a PC from scratch and installing an operating system.



The Glossary Items you were looking for.

BIOS

LearnPC



The Keywords you were looking for.

AGP

AGP

User's Guide

Users Guide to the Presentation Manager

The Presentation Manager (or "PM") is a browser-based program to organize and display modules on a topic. The following functions may be accessed through this program:

- A *Module Listing* providing links to each of the modules.
- A Java applet that allows access to *Module Select*, a consolidated *Glossary*, a consolidated *Keyword* listing, and a description of the PM’s functions and procedures in an *About Box* that can be accessed at any time.
- A link within a module to *Module Specific Glossary* words that link directly to the specified word.

Directions for Use

Getting Started

The first time that you try to use the PM you may be asked to download a Java plug-in. This is because the browsers out at this time do not have the Swing components loaded. This plug-in is simply to give your browser access to these components. There are two possible ways for this download to work. The whole process may be automated for you, in which case you will only have to click on the wizard that appears and the plug-in will be installed on your computer with no other work on your part. The other possibility is that you are taken to the Sun web page that houses the Java 1.2.2 download. If this is the case, scroll down until you find the link that says, “Download”, save the downloaded file to your desktop, and once the file is downloaded, minimize your browser and execute the file. Once this is done, you will never have to download this plug-in again.

Basic Display and Link Info

Upon the system properly loading, you will see an octopus in the top right hand corner of the window. Many of the functions of this program live in that little picture nicknamed "Nettie the Octopus". To access these functions click on Nettie, and a Java applet screen will appear with 4 tabs: *Module Select*, *Glossary*, *Keywords*, and *About Box*.

Next to Nettie is a word saying "Back". This is a link that will always take you to the main menu - or Module Listing - which is described below.

The rest of the top of the window should have the title “NetLab Presentation Manager.” In this location should be the title of whatever module you are in.

Below the title taking the largest section of the window is the main menu, otherwise called the Module Listing.

Module Listing

On the main portion of the window originally loaded is the Module Listing. This is the main menu of loaded modules and provides a link to all of the loaded modules and a short text about each. The name of the module is the link to the opening page of the module. Click on the name to link to a module.

Module Specific Glossary

Within each module are words that are important to that module. These words have a short definition that can be accessed by clicking on the words. If you see a word that looks like a link to another page, click on it. It will either take you through to another page in the module (if it is not a glossary word), or it will open up a small window showing the definition of the word. You do not need to close this window; you may wish to click on the main window and just continue on with the module. The next glossary word on which you click will automatically bring the glossary window into focus again.

Main Applet

After clicking on Nettie, the main applet appears. In this applet are 3 main areas: the tabs, the viewing area, and the top boxes. On the left side of the applet window are 4 tabs: *Module Select*, *Glossary*, *Keywords*, and *About Box*. These determine whether the viewing portion of the applet is displaying module information, glossary words, keywords, or text about the PM. Each tab will be covered separately.

On the top of the applet window are 4 boxes: a text box, a drop-down box, and two buttons. The drop-down box will always display the list of modules. Clicking on one of these will always take you to the Module Select tab and show the text about the selected module. The first box is a text box to type in a word or portion of a word for which to search. The second box is the “Search” button that initiates the search for the indicated item. For what is searched depends upon what tab is activated, so will be described in the individual tabs. The “Show All Items” button will show the same items as the “Search” button, but shows the entirety of the list and not just selected items.

Module Select

The first tab is Module Select. The main portion of the applet shows text about the module that appears in the drop-down box on the top of the applet. This module can be accessed by clicking on the “Open Module” button on the left side of the *about* text.

The search within this tab will be for the glossary words. So by typing in a word and hitting “Search” or by hitting the “Show All Items” button, you will be taken to the “Glossary” tab and will be shown glossary terms.

Glossary

The second tab is for glossary terms. By typing in a word and hitting “Search” or by hitting the “Show All Items” button, you will be shown glossary terms.

Between the term and the definition is a link to the first page of the module from which the definition originates. Therefore you may click on the link to access the module to further research the term. If there is more than one module using a term, the term will appear only once, with the module links and definitions listed beneath it in order of the module listing.

Keywords

The Keywords tab allows, by typing in a word and hitting "Search" or by hitting the "Show All Items" button, access to the list of keywords for all loaded modules. Each keyword will indicate to which module it is linked, and will provide a link to the programmed location within the module for that term.

About Box

This is text describing the functions and procedures of the PM. It also provides a link to text on how to create new modules.

The search within this tab will be for the glossary words. So by typing in a word and hitting "Search" or by hitting the "Show All Items" button, you will be taken to the "Glossary" tab and will be shown glossary terms.

Evaluation of the System

Proposed enhancements:

Things that should be fixed:

Change the LayoutManager for PMembedable so that the window is resizeable. PmEmbedable is the applet that runs in the corner of the browser.

Find a new way to get the list of modules without information in the modules.txt file. Presently the modules.txt file has to be edited each time a module is added.

More robust string parsing. In particular the work done by the Loader class is crude. A XML DTD could be developed that allows for the use of java XML code.

The searching method is crude. It does a search word.toLowercase() against the list of terms and if that fails it searches on the first three letters of the word. The searching needs to be more feature full.

The keyword presentation is ugly and should be fixed.

The gif the Button to display the PMembedable should be dynamically loaded. Animated? This would allow for more customization of the system without recompiling the code for the Presentation Manager.

Idea for version #2

Make a custom web browser out of Java that runs the that has the PM fully embedded in it.

Increase speed of loading. (faster processor)

Change the search in the main applet when the Module Select tab is selected to be through the module level keywords, no through the glossary words.

Testing Plan

There are two major components to test, the PM and the modules. Therefore there is a test plan for each.

PM testing

The portions of the PM that need to be tested are:

- loading all appropriate module data, including the module's about file
- loading all keyword information with proper link information
- loading all glossary information with definitions stored properly
- sorting the glossary appropriately for display and combining like glossary term definitions under one heading
- ensuring the glossary definitions have a link to the appropriate module
- sorting the keywords appropriately with the links working correctly
- searching and displaying the proper glossary and keywords
- displaying the module list with the text about the module properly, and having a working link to the indicated module
- ability to return to the module list after being in a module
- ability to exit and reenter PM in the same browser session (known bug)
- ensuring that the PM lives on while a module is in use
- ability to reload the PM after adding or taking away a module with the new information intact

To test, the following should be done:

Create and save in the *modules* directory a modules.txt file with four lines in it (according to the spec, there must be a blank line at the end of the file).

module1
module2
module3

Create three folders in the *modules* directory named the same as the three lines above. Within each of those folders have an about.htm file, glossary.htm file, and keywords.htm file formatted properly according to the specs.

Each *about* file should say "This is the about file for moduleX" with the X replaced by the proper number. The reason for this is so it is very evident where the *about* file came from so as to easily check the module listing.

Each glossary should have three terms and definitions.

module1:

1) Common Word - This is a word from module1 common to all modules.

2) Stranger - This word from module1 is in no other module, but is close to a word in another module.
3) Dohicky - This word from module1 is in no other module.

module2:

1) Common Word - This is a word from module2 common to all modules.
2) Strange - This word from module2 is in no other module, but is close to a word in another module.
3) Do Over - This word from module2 is in no other module.

module3:

1) Common Word - This is a word from module3 common to all modules.
2) Strong - This word from module3 is in no other module, but is close to a word in another module.
3) Doomaflodgit - This word from module3 is in no other module.

The reasoning behind these words are thus:

- Common Word to check that all of the definitions are displayed properly when more than one module has the same definition. Also to check a term with more than one word.
- Dohicky, Do Over, and Doomaflodgit are to check the search algorithm; they only have the same first two letters and should not appear in the same search.
- Strong, Strange, and Stranger have the same first 3 letters. They should come up separately when requested specifically, but should both come up when "Str…" is entered.

Each *keyword* file should have two entries: one as a "<modkey>", a module level keyword, which points to "content/index.htm"; the other pointing to an anchor on a second page of each of the test modules. The keyword for two of the modules should be the same so as to test the output of identical keywords. The search algorithm used is the same function as what is used in the glossary search, so does not need to be tested separately.

Inside each test module should be a *content* directory that has two files in it. The first is an index.htm file that simply says "This is the *index* file for moduleX", replacing the X with the proper number, and a link to "page 2." The second file should have 20 numbers running down the left side. At number 10 put an anchor that will be used in the *keyword* file above.

Now once all of this is in place the following can be done to actually test all of the PM's functionality.

Load the PM. See if it looks similar to this:

module1

This is the about file for module1.

module2

This is the about file for module2.

module3

This is the about file for module3.

Click on all of the links and make sure you get to each of the modules' *index* files.
On the first module, hit the back button on the browser to reenter the module listing. On the second module, hit the link back to the module listing that is next to Nettie the Octopus. On the third module go into the second page of the test module and hit the link next to Nettie.

If these steps come up with the expected results, close the browser. Go into the modules.txt file and take out "module2", leaving only module1, module3, and the blank line at the end. Bring back up the browser and reenter the PM. See if only module1 and module3 are shown. Also click on Nettie, view the full glossary and make sure that only module1's and module3's words are displayed.

If these steps come up with the expected results, close the browser again, go back to the modules.txt file, reenter "module2" after "module3", so that modules.txt looks like

```
module1
module3
module2
```

including the blank line. Reenter the PM and see if the modules are listed: 1, 3, 2.

Testing of the module listing and the module links are complete. Now to test the glossary and keywords.

Click on Nettie. Pull up the full glossary and check to see that all of the words are there in alphabetical order. Also check to make sure that all entries with identical words ("Common Word", and "Dohicky") have the term only appear once, but all of the appropriate definitions and links appear. Click on a link to module1. Access the full glossary again while in the module's index page. Click on a link to module2. Go to page 2 of module2 and access the full glossary. Click on a link to module3.

If these steps come up with the expected results, search within the glossary words for "Strang". You should get "Strong", "Strange", and "Stranger" (the search algorithm only checks the first three letters if no specific word is found). Type in "Strange" and you should only get "Strange". Type in "Doh" and you should only get "Dohicky". Type in "Do" and you should get nothing. Type in "Do " (with a space after Do) and you should only get "Do Over". Type in "Common" and you should get all three definitions of "Common Word".

Finally go to the keyword portion of the PM and check all of the links, ensuring they lead to the appropriate location.

If all of these tests pass, the PM should be fully functional.

Module Testing

Module testing is really up to the writer of the module, but a module writer needs to test the following to ensure that the module will work properly. The necessary tests are:

- that all of the links are done correctly, starting with the index page through the rest of the expected path.
- that all of the glossary word links within the module open up a new window with the appropriate word visible.
- that the keywords are connected to the proper location.
- that the proper title appears upon loading the module.

One other thing to check is that all links have the correct capitalization. A Window's platform will not differentiate, but other operating systems will.

To test, the following should be done <u>before</u> putting the module name in the modules.txt file (before it is visible to the PM), but after the module has been made to fit the specs.

> Pull up the index.htm file on a browser. Click on every link to every other page. Make sure the expected page comes up. Once page linkage is verified, click on every glossary word to ensure that the proper word appears in the glossary window.
>
> If these steps come up with the expected results, open the keywords.htm file up in the browser. Click on all of these links to make sure that they all link to the appropriate page and location. You may have to take out the <modkey> tags to see those words, but remember to add them back in before going on.

If all of the links within the module seem to be correct, add the module to the *modules* directory, and add the name of that directory to the modules.txt file.

> Pull up the PM and see if the module you added is listed in the module listing. Check the link to make sure the initial page comes up. Also check to make sure the title frame has changed to the appropriate text. Pull up the glossary and spot check to make sure your words are available. Try one link from the glossary to your module. Pull up the keyword listing and spot check to make sure your words are available. Try one module level link and one regular keyword link.

This concludes the necessary tests for a new module.

Code Listing

Full source available at:
http://www.evergreen.edu/sos/1999-2000/projects/nsl/docs.html

dbWrapper

```
 /**
  * This Class encapsulates the creation of all of the data structures used by
 * a Presentation Manager Front end.
 * Creation date: (04/13/00 23:55:10)
* @author: Erik Ordway
 The Presentation Manager

The NetLab Project of The Evergreen State College

Java code
Copyright ©  2000 by Erik Ordway crainm@techie.com

Non Module specific html, css, and graphics
Copyright ©  2000 by Brook Hatch

Module specific html, css, and graphics
Copyright ©  2000 by the Authors of the module

Welcome to The Presentation Manager. This document covers is the licenses of The
Presentation Manager Release 4.0.  This program and its accompanying modules are a
work in progress and are the work of many individuals. If you are interested in
helping with this project, please contact the NetLab project at
http://grace.evergreen.edu/nsl.  The latest version of this program is always
available from the NetLab Wide Web server. It may also be downloaded in a variety
of formats and compression options from the NetLab web server or run from that
site.

Redistribution and use in source Java code, html, css, and graphics  and 'compiled'
forms Java bytecode with or without modification, are permitted provided that the
following conditions are met:

Redistribution of source code Java source code must retain the above copyright
notice, this list of conditions and the following disclaimer as the first lines of
this file unmodified.

Redistribution in compiled form Java bytecode with accompanying modules and support
html, css, and graphics must reproduce the above copyright notice, this list of
conditions and the following disclaimer in the documentation and/or other materials
provided with the distribution.

Important: THIS PROGRAM AND ITS ACCOMPANING MODULES ARE PROVIDED BY THE NETLAB
PROJECT AND ITS AUTHORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE NETLAB PROJECT BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION,
EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
import java.io.*;
import java.net.URL;
import java.net.MalformedURLException;
import javax.swing.*;
import javax.swing.text.html.*;
import javax.swing.SwingUtilities.*;
import javax.swing.event.*;
import javax.swing.text.*;
import java.util.Enumeration;
import java.util.Vector;
import java.lang.String;
import javax.accessibility.*;
import java.applet.*;
class DBWrapper
{
      protected Loader load = new Loader();
      private Glossary_db glossaryDB = new Glossary_db();
      private Keyword_db keywordDB;
      protected Modules_db modules;
      protected java.applet.AppletContext context;

/**
 * Wrapper constructor comment.
 */
public DBWrapper() {
      super();
}
/**
 * Wrapper constructor.  This action of the constructor is done by buildData().
 */
public DBWrapper( java.applet.AppletContext context, URL codebase ) {
      super();
      setContext(context);
      buildData(codebase);
      return;
}
/**
 * This method contains all the data structures use by a front end such as
      PMembedable.
 * an instance of the class Loader() is created and then called to fill the data
      structures
 * Creation date: (04/13/00 23:59:44)
 */
private void buildData(URL url)
{ //Setting context with the applets browser context



      //run the modules list from Loader()
      load.loadModules(url.toString());
      int i = 1;
      setModules();
      //end load modules list

      //begin load Glossary
      context.showStatus("Loading Glossary information from Modules");
      glossaryDB = new Glossary_db();
      load.loadGlossary(glossaryDB);
      load.loadHeadNTail(url, glossaryDB, "modules/glossary.htm");
      //end load Glossary

      //begin load Keyword
      context.showStatus("Loading Keyword information");
      keywordDB = new Keyword_db();
      load.loadKeyword(keywordDB);
      load.loadHeadNTail(url, keywordDB, "modules/keywords.htm");
      //end load Keyword


      context.showStatus("All information has been loaded");
      load = null;
}
/**
 * Insert the method's description here.
 * Creation date: (04/14/00 00:06:00)
 * @return java.applet.AppletContext
 */
public java.applet.AppletContext getContext() {
      return context;
}
/**
 * Insert the method's description here.
 * Creation date: (04/14/00 00:06:00)
 * @return Glossary_db
 */
public Glossary_db getGlossaryDB() {
      return glossaryDB;
}
/**
 * Insert the method's description here.
 * Creation date: (04/14/00 00:06:00)
 * @return Keyword_db
 */
public Keyword_db getKeywordDB() {
      return keywordDB;
}
/**
 * Insert the method's description here.
 * Creation date: (04/14/00 00:06:00)
 * @return Loader
 */
public Loader getLoad() {
      return load;
}
/**
 * Insert the method's description here.
 * Creation date: (04/14/00 00:06:00)
 * @return Modules_db
 */
public Modules_db getModules() {
      return modules;
}
/**
```

```
 * Insert the method's description here.
 * Creation date: (04/14/00 00:06:00)
 * @param newContext java.applet.AppletContext
 */
protected void setContext(java.applet.AppletContext newContext) {
      context = newContext;
}
/**
 * Insert the method's description here.
 * Creation date: (04/14/00 00:06:00)
 * @param newGlossaryDB Glossary_db
 */
private void setGlossaryDB(Glossary_db newGlossaryDB) {
      glossaryDB = newGlossaryDB;
}
/**
 * Insert the method's description here.
 * Creation date: (04/14/00 00:06:00)
 * @param newKeywordDB Keyword_db
 */
private void setKeywordDB(Keyword_db newKeywordDB) {
      keywordDB = newKeywordDB;
}
/**
 * Insert the method's description here.
 * Creation date: (04/14/00 00:06:00)
 * @param newLoad Loader
 */
private void setLoad(Loader newLoad) {
      load = newLoad;
}
/**
 * Insert the method's description here.
 * Creation date: (04/14/00 00:06:00)
 * @param newModules Modules_db
 */
protected void setModules() {
      modules = load.getModules();
}
}
```