

A USER-FRIENDLY INTERFACE TO RCS
AND ITS USE AS A SOFTWARE REPOSITORY

By

SUNIL CHAKRAVARTHY NADELLA

Bachelor of Engineering

Venkateswara College of Engineering

Madras, Tamil Nadu, India

1990

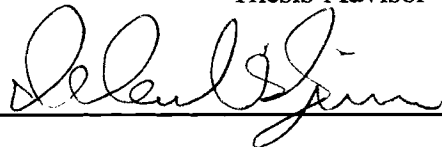
Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May 1995

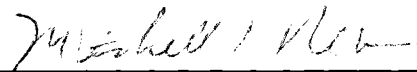
A USER-FRIENDLY INTERFACE TO RCS
AND ITS USE AS A SOFTWARE REPOSITORY

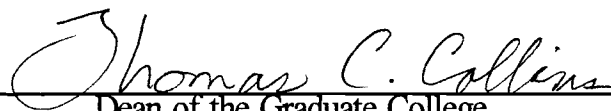
Thesis Approved:



Thesis Advisor







Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Dr. Mansur H. Samadzadeh for his intelligent guidance, constructive criticism, valuable suggestions, and his strong interest in the progress of my thesis. I also wish to thank Drs. Benjamin and Neilsen for serving on my committee.

I express my gratitude to my parents for their encouragement and moral support. I wish to thank my brother for his friendly encouragement. Finally, I want to thank my wife, Lata, for her patience and understanding during the thesis preparation. Her assistance in testing the package and in preparing the manuscript was invaluable.

I express my gratitude to my beloved lord Bhagavan Sri Sathya Sai Baba without whose grace all this would not have been possible. I dedicate this thesis to my beloved Bhagavan.

TABLE OF CONTENTS

Chapter	Page
I INTRODUCTION	1
II LITERATURE SURVEY	5
2.1 Version and Configuration Management Systems	5
2.1.1 Source Code Control System	6
2.1.2 Revision Control System	10
2.1.3 DOMAIN Software Engineering Environment	13
2.1.4 GANDALF Project	14
2.2 Software Reuse	16
2.2.1 Introduction	16
2.2.2 Composition Technology	16
2.2.2.1 Modification and Combination of Modules	18
2.2.3 Generation Technology	20
III DESIGN AND IMPLEMENTATION ISSUES	22
3.1 Design	22
3.2 Implementation	25
3.3 User Interface	28
3.4 Implementation Platform and Environment	40
IV SAMPLE SESSION AND OBSERVATIONS	42
4.1 Sample Session with VCI	42
4.2 Observations and Limitations	46
V SUMMARY, CONCLUSIONS, AND FUTURE WORK	54
5.1 Summary	54
5.2 Conclusions	54
5.3 Future Work	54
REFERENCES	56

APPENDICES	59
APPENDIX A- GLOSSARY AND TRADEMARK INFORMATION	60
APPENDIX B- USER GUIDE	62
APPENDIX C- LIBRARY ADMINISTRATOR GUIDE	67
APPENDIX D- PROGRAM LISTING	73

LIST OF FIGURES

Figure	Page
1. Logical structure of versions	2
2. Chain of deltas in SCCS	7
3. Reverse deltas in RCS	11
4. Forward deltas in RCS	11
5. Hierarchial structure of components in the GANDALF project	15
6. Initial screen	29
7. Screen with drawn buttons	30
8. Description dialog	32
9. List dialog	34
10. Pattern search dialog	36
11. Thesaurus dialog	39
12. Screen with drawn buttons	43
13. Description dialog of draw.c	44
14. List dialog of draw.c	45
15. Zoom-out mode of the display from the Figure.c file	47
16. Zoom-in mode of the display from the bubblesort.c file	48
17. Zoom-out mode of the display from the bubblesort.c file	49
18. Space utilization graph	50

Figure	Page
19. Percentage space utilization graph	51
20. Structure of the main.dat file	69
21. Structure of the experimental.dat file	70
22. Structure of the update.dat file	70
23. Structure of the link.dat file	71
24. Structure of the thesaurus.dat file	71

CHAPTER I

INTRODUCTION

The facility to regulate and administer modifications to a software module is called configuration management [Babich86]. There can be several variants of a software module. The reason for this is that there are defects to be rectified, new features to be incorporated, or the software module has to be adapted to new environments [Rochkind75]. The variants or the versions of a software module are usually stored in separate files.

Consider two versions of a software module, one for the Sequent DYNIX/ptx operating system and one for the Sun operating system. If a bug is discovered in the version for the Sun operating system, the same bug may prevail in the version for the Sequent DYNIX/ptx operating system. It is not enough to fix the version for the Sun operating system; the version for the Sequent DYNIX/ptx operating system must also be fixed. The modification(s) made to one version must be made to all versions of the software, if the versions are stored in separate files. Due to negligence, some versions may be left out without the modification(s). This problem of updating of all versions is solved in many software configuration management systems like SCCS [Rochkind75] by storing the versions as modules as shown in the Figure 1.

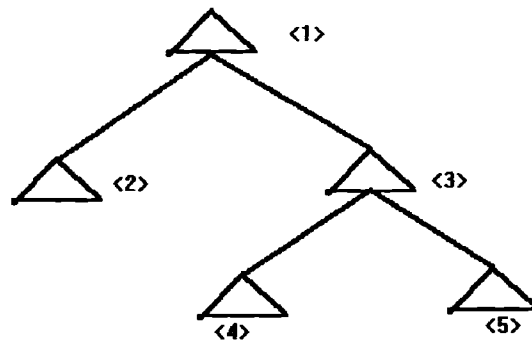


Figure 1. Logical structure of versions

Let us assume that the version for the Sun operating systems is made up of the modules <1>, <3>, and <4> and the version for Sequent DYNIX/ptx is made up of the modules <1>, <3>, and <5>. The notation used to represent the modules in this report is similar to the notation used by Miller, et al. [Miller89]. If a bug is located in module <1> or module <3>, it needs to be fixed only once in the file associated with module <1> or <3>. This correction is incorporated into the other version as the other version also depends on the same file.

The second problem associated with the storage of versions of a software module in separate files is the potential waste of storage space. It is usually the case that versions do not vary much from one another. Therefore, storing a software module again in its entirety with slight modifications in a separate file is a waste of storage space. For example in Figure 1, if one version consists of modules <1>, <3>, and <5> and the other version is made up of modules <1>, <3>, and <4>, and if the two versions are stored separately in two files, both modules <1> and <3> are stored in two different files which is a waste of storage space. This problem of possibly wasting storage space is solved in

many configuration management systems like SCCS [Rochkind75] in the following way. A single version is stored completely and other versions are stored based on their difference from this version. This difference file is called a delta [Babich86]. The delta for the DYNIX/ptx operating system has control characters to remove module <4> and insert module <5>. Therefore, modules <1> and <3> are stored only once, resulting in significant saving in storage space.

When a number of programmers are accessing a single software module, the changes made by one programmer may interfere with the progress of other programmers. If a bug is introduced by a change to a module, all programmers have to wait till it is located and fixed. The software module may suddenly stop behaving as required. This may be because the changes incorporated by one programmer to introduce a new functionality interferes with a function programmed by another programmer. This oversight on the part of one programmer may be because that programmer was not aware of the implementation details of the other function. Under these circumstances, it is desirable to remove the recently-introduced modifications, restoring the software to its original state for other programmers. Meanwhile, the modification can be debugged and introduced again.

If many modifications are introduced before the bug is discovered, the modification that introduced the bug must first be identified and then removed from the software module. The process of identifying and eliminating the modification that introduced the bug is simplified in configuration management environments like SCCS [Rochkind75] in the following way. Each modification to the software module is stored in a separate file. The difference between the version of the software module without the modification and

the version with the modification is the delta of that version. Therefore, the delta is nothing but the modification. To identify the modification that introduced the bug, the deltas corresponding to recently-introduced modifications are undone one by one and the delta giving the error is isolated. To remove this isolated modification, all that needs to be done is to erase the delta corresponding to that modification. Now other programmers can work on adding their own deltas or modifications while the defective modification is rectified.

Another advantage of storing the modifications as deltas is that a modification done by one programmer can easily be identified by other programmers by going through the delta, in contrast to going through the entire software module. To keep track of the modifications done to a software module, configuration management systems like SCCS [Rochkind75] stamps each delta with the author's name, date the modification was made, and the reason it was made. This maintains the history of the software module.

By integrating the concepts of version and configuration management with the composition technologies of software reuse, an efficient software repository can be built. This is the theme of this thesis. In Chapter II, the various version and configuration management techniques are reviewed. An overview of software reuse with an emphasis on composition technology is also presented. The design and implementation issues involved in creating the interface to the software repository, called the Version Control Interface (VCI), are discussed in Chapter III. The evaluation of the tool is presented in Chapter IV. Finally, the conclusion and future work are presented in Chapter V.

CHAPTER II

LITERATURE SURVEY

2.1 Version and Configuration Management Systems

Strictly speaking, version management and configuration management refer to slightly different notions. Version management is the ability to organize the successive "forms" of a software module. Version management manages only individual software modules. Configuration management is the capacity to control the association of versions of modules in composing software systems.

Currently, there are two general approaches to dealing with versions in software development environments. One is to provide configuration management on top of version management, and the other is vice versa. In this section, two version management tools, RCS [Tichy85] and SCCS [Rochkind75], are reviewed. Then, the DOMAIN software engineering environment [Leblang85], which is representative of the first approach of providing configuration management on top of version management, is reviewed. Finally, the GANDALF project [Miller89], which follows the second approach of providing version management above configuration management, is reviewed.

2.1.1 Source Code Control System

The source code control system (SCCS) stores each change to a software module in a separate delta [Rochkind75]. The deltas from each change form a chain. When a software module is coded and placed under SCCS, it is said to be at release 1 level 1 (1.1). If the module is required for changes to be made, it must be checked out from SCCS. After the changes are made, when the module is checked in, the changes are stored as a delta. Each delta or change represents a new level. When a new release is planned, the release number of the next delta can be incremented and the level can be reset to 1. Even after starting a new release, deltas can be added to an old release at the end of the corresponding chain. Deltas cannot be added between two deltas.

There are two special purpose deltas, the optional delta and the inclusion/exclusion delta. The optional delta is applied only when the option letter (any alphabetic character specified by a programmer while creating the delta) that is attached to it is cited. The option letter is stored along with the delta, and the delta is applied only when the user specifies its inclusion by specifying the option letter as a command line option while checking out the file corresponding to a higher version. This delta is used to make temporary changes or to include user preferences. If a new version is made in an old release, it will not be introduced in later releases that had already started before the new version was made.

For example in Figure 2 (adapted from [Rochkind75]), if a delta is made at release 1, consisting of modules <1.1> through <1.4>, then it won't be applied to release 2, consisting of modules <2.1> through <2.3>, because release 2 was started before the delta

was made. Therefore, a different delta has to be made at release 2 to include the effect of delta <1.5> in release 2. This delta, similar to delta <2.4> in Figure 2, is called an inclusion delta. If in the example of Figure 2, the change incorporated by applying delta <1.3> is found to be incorrect, then instead of deleting delta <1.3>, another delta, say <1.5>, can be introduced to undo the effect of delta <1.3>. Delta <1.5> is called an exclusion delta. This delta is necessary because valuable development information of the software module may be lost otherwise by the deletion of an existing delta.

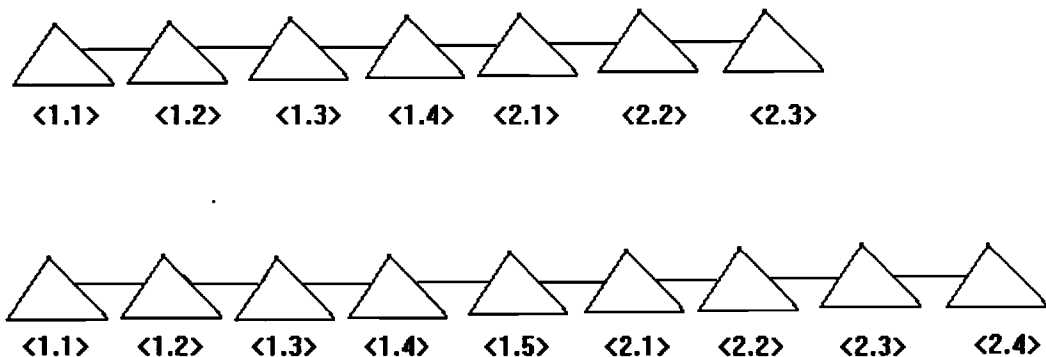


Figure 2. Chain of deltas in SCCS

Changes to software modules or deltas are recorded as insertion or deletion of lines. If even a single character is changed, the line is marked as deleted and a new line with the modified character is marked as inserted. The software module and its deltas are stored in one sequential file. This file has two parts, the body, where deltas are stored, and the control record that specifies the effect of deltas. If a line in version x of a software module is changed, it is enclosed by two control records. The "begin deletion"

control record has the version number from which the succeeding lines till the "end deletion" record must be deleted. Immediately after this, the modified line, again enclosed by two control records, is inserted. The "begin insertion" control record has the version number into which the following lines have to be inserted. The "end insertion" control record marks the end of the insertion. When the source code corresponding to version x has to be made, each line in the sequential file is inspected. When a "begin insertion" or a "begin deletion" control record is encountered, an entry is made into a control queue. An entry in the control queue has one of the following values: "yes", "no", or "NULL". Only if the value is "yes" is the toggle switch set. If the value is "NULL", the next control record is inspected. When each line in the sequential file is inspected, the toggle switch is examined, and only if it is set is the line retained. When an "end insertion" or an "end deletion" control record is encountered, the corresponding entry in the control queue is deleted.

SCCS restricts access to versions of a software module to only those whose login id is present in the access group. The second type of protection provided by SCCS is to control the modification of the versions. Some versions can be locked out for revision. SCCS then restricts the checking in of a revision of that version. This supports the idea of software baselining, according to which a baseline is set up and none of the modules within the baseline can be modified. This is important for the future development of the software system because the stability of the baselined modules is ensured.

SCCS provides on-line documentation in the deltas. SCCS stamps each delta with the name of the user who makes the change, time and date the change was made, and the

reason the change was made. The user is prompted for the reason for the change when checking in the change. Therefore, the accuracy and usefulness of the reason recorded depend on the sincerity of the user.

Operations supported by SCCS are listed below [Rochkind75].

admin	create and administer SCCS files
cdc	change the delta commentary of an SCCS delta
comb	combine SCCS deltas
delta	make a delta (change) to an SCCS file
get	get a version of an SCCS file
prs	print an SCCS file
rmDEL	remove a delta from an SCCS file
sact	print the current SCCS file editing activity
sccsdiff	compare two versions of an SCCS file
sccsfile	give the format of the SCCS file
sccstorcs	build RCS file from SCCS file (see subsection 2.1.2)
unget	undo previous "get" of an SCCS file
val	validate an SCCS file
what	identifies an SCCS file

The val operation can be used to determine if an SCCS file conforms to the characteristics specified in the arguments. The arguments that can be specified are SID (SCCS identification string), the SCCS keyword %m% called name, and the keyword %y% called type. If an SID is given as an argument, the file is checked to see if the SID exists and if it is valid. If the name or type keyword is given in the argument, it is compared with the corresponding keyword in the file and an error message is printed if they don't match.

Since a software module and its various deltas are stored in one sequential file, the time taken to make a version is the same irrespective of the number of deltas applied. For example in Figure 2, it takes the same amount of time to make the file corresponding to version <1.2>, which involves applying only <1.1> and <1.2> on the sequential file, as

the amount of time taken to make the file corresponding to version <2.3>, which involves applying <1.1> through <2.2> on the sequential file. This disadvantage of SCCS is overcome by RCS [Tichy85] which is described in the following subsection.

2.1.2 Revision Control System

The revision control system (RCS) manages versions of text files [Tichy85]. RCS stores all old versions of a software module in a space-efficient way. RCS maintains a complete history of a software module by tracking all changes automatically. RCS records the date, time of change, author, and reason for the change. RCS maintains the integrity of the software module by allowing only one programmer to check out a version for revision. If a version is already checked out, other programmers can check out the version for reading purposes only. RCS maintains a tree of the versions. The versions of a software module can be assigned symbolic names, and the state (a variable associated with a version of a software module to indicate the status of the version) can be assigned values such as released, stable, and experimental, which can help in configuring the software module.

RCS, like SCCS described in Subsection 2.1.1, stores changes as deltas. Deltas are line based. Therefore, a change of even one character in a line is stored as a deletion of the line and the inclusion of the new modified line. To decrease access time, RCS stores the most recent version fully as the base version. All other versions are stored as reverse deltas. Reverse deltas are the modifications that must be made to a version to produce the files corresponding to the previous versions as compared to regular deltas which are

modifications made to versions to produce the files corresponding to the later version, as illustrated in Figure 3 (adapted from [Tichy85]).

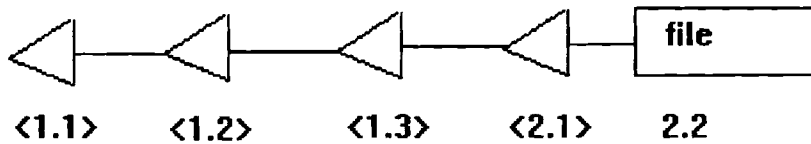


Figure 3. Reverse deltas in RCS

Since the latest version is the one most frequently accessed, it can be extracted quickly because of this method. RCS uses forward deltas to store deltas in branches. That is, branch deltas are stored as differences from their root. For example in Figure 4 (adapted from [Tichy85]), delta 1.2.1.1 and delta 1.2.1.2 are stored as forward deltas of 1.2 and 1.2.1.1, respectively.

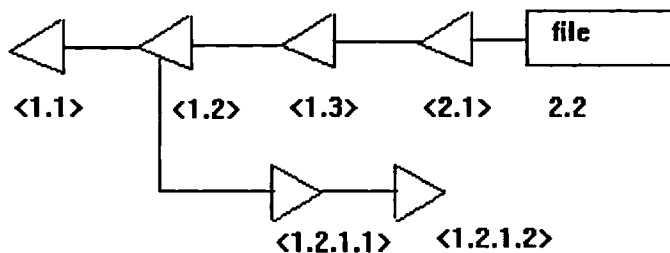


Figure 4. Forward deltas in RCS

In SCCS, as pointed out previously in Subsection 2.1.1, all deltas are stored in one sequential file. To make a particular version, the entire file is scanned. Depending on the control characters, a particular line is kept or left out. Thus, the time taken to make any

version does not change, irrespective of the number of deltas applied.

RCS uses a faster algorithm to build the file corresponding to a version. It works as follows. In RCS all operations are carried out on a one-dimensional array called the piece-table. This array contains the starting position of all the lines in the sequential file (that is, the RCS file) to be included in the file corresponding to that version. Therefore, using this array and by indexing into the sequential file (and copying the corresponding lines into a new file), the file corresponding to that version can be made efficiently. For example, the piece-table corresponding to version <1.1> of a module has the starting positions of all the lines in the initial sequential file, as it is the first version. When version <1.2> of a module is created, the piece-table is updated. To make the file corresponding to version <1.2>, the sequential file need not be accessed line by line as in SCCS, rather the lines indicated by the piece-table are included in the file. The efficiency of this algorithm is realized when the number of deltas applied is fewer than the number of deltas stored. The time taken to make a file corresponding to a version depends on the size and the number of deltas applied.

When a version is checked out from RCS and locked, other programmers can check out that version for inspection only. They cannot check in a delta on that version. The lock on a version can be released only by the person who has checked it out. If a lock on a version is broken, an e-mail is automatically sent to the person who has locked out the version.

The selection of a version can be done by the version number, cut-off date, or author's name. An important selection method that is permitted in RCS is selection by a

symbolic name. In this method, a symbolic name is associated with some deltas, and these deltas can be selected using the symbolic names. The symbolic names are then mapped onto the numeric version numbers. The operations supported by RCS are listed below [Tichy85].

ci	check in versions
co	check out revisions
ident	extract identification markers
rccs	change RCS file attributes
rccs-clean	clean working directory
rccs-diff	compare versions.
rccs-freeze	freeze a configuration
rccs-merge	merge revisions
rlog	read log messages

2.1.3 DOMAIN Software Engineering Environment

The DOMAIN Software Engineering Environment (DSEE) is built on top of SCCS.

It consists of the following major functionalities [Leblang85].

History Manager. The history manager keeps track of the development history of the software modules. DSEE uses a database management system, called D3M, for this purpose.

Configuration Manager. The configuration manager builds programs from versions. When a software module has to be rebuilt from the changes in the files on which it depends, the configuration manager performs the construction as needed.

Release Manager. The release manager saves the configurations of the software modules that are released.

Task Manager. The task manager detects the changes made to a software module

anywhere on the network the software is developed on. This is needed to extend the software development efforts across a network and not to restrict configuration and version management to single machines.

Monitor Manager. The monitor manager keeps track of user-defined dependencies. When the corresponding files are changed, the user is alerted.

Advice Manager. The templates for redoing common tasks are stored in the advice manager. It also holds information about the software being developed.

DSEE uses SCCS to provide source code control. To build a particular system, the specification of a system model is the first step. A system model has information about the components of the software modules that go into the system and their dependencies. The second step in building a system is to specify a configuration thread. While a system model specifies the modules that compose a system, a configuration thread specifies the versions of the modules that make up the system. Using a system model and a configuration thread, a system can be built.

2.1.4 GANDALF Project

In the GANDALF project, version management is built on top of configuration management [Miller89]. The source code components are stored in hierarchical structures. Consider Figure 5 (adapted from [Miller89]). Let the two versions of a software system, one made up of modules <1>, <2>, <3>, <4>, and <5>, and the other made up of modules <1>, <2>, <3>, <4>, and <6>, be for two separate machines as indicated in Figure 5.

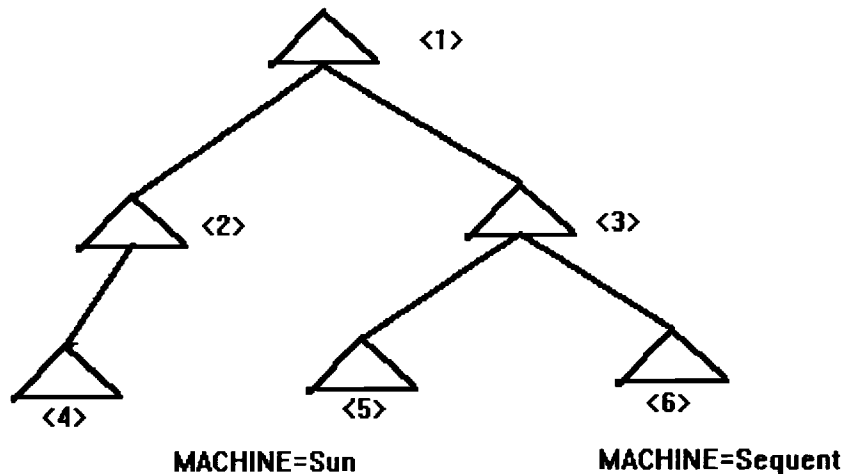


Figure 5. Hierarchical structure of components in the GANDALF project

Configuration management in the GANDALF project is achieved as follows. Each component is tagged with an <attribute, value> pair. If the value of an attribute in a component is not specified, that attribute can take all of the possible values. For example in Figure 5, the attribute can be "MACHINE" and the possible values can be either "Sequent" or "Sun". The value for "MACHINE" is not specified in modules <1>, <2>, <3>, and <4>, thus making them suitable for both machines. The software system corresponding to a particular version is made by inspecting each module from the root and choosing only those modules whose attribute values match a predefined value.

Version control on top of configuration management is achieved as follows. Each version consists of all the modules making up a software system. When a module in a software system is modified, a new version of the software system is made by combining the modified version of the module with the other unchanged modules.

The advantage of this system is that the development path of a software system is

captured. Using this path, a software system can be rolled back to its previous state if an erroneous change is made.

2.2 Software Reuse

2.2.1 Introduction

Software reuse is the process of creating software systems from existing software rather than from scratch [Krueger92]. The major motivations and justifications for software reuse can be listed as an increase in productivity by a reduction in the amount of code written, a reduction in the amount of documentation and testing required, and an increase in software quality due to the reused software being already well-designed, documented, and tested [Tracz87].

From a technological viewpoint, reuse of software can be divided into two classes. One class employs composition technologies and the other employs generation technologies [Biggerstaff89]. These two notions are described in the next two subsections.

2.2.2 Composition Technology

In composition technologies, software systems are built from existing modules with little or no modification. The major issue of the composition technology is the maintenance of the requisite software module library. McIlroy proposed the idea of forming a software components catalog from which software parts could be selected [McIlroy69]. Prieto-Diaz used the faceted method of classification from library science to classify software modules [Prieto-Diaz89]. Software modules can be described by their

attributes such as the functions they performed, the way they perform them, and their implementation details.

The main characteristics of the modules used are function, object, medium, system type, function area, and setting. These characteristics are briefly described below.

Function: The action performed by the software module.

Object: The items manipulated by the module, for example, variables, characters, or lines.

Medium: The entities that serve as the "locale of action", such as files and tables.

System Type: Functionally identifiable modules, e.g., a report formatter.

Function Area: The terms describing the area of application of the software modules, e.g., a purchasing department or a sales department.

Setting: The environment where the application is going to be used, e.g., a chemical plant or a print shop.

A typical example of the description of a module would be <compress, files, disk, file-handler, DB-management, catalog-sales>.

There is a need for a controlled vocabulary while naming the above characteristics in a description, as there is more than one way to describe the same characteristic. For example, move and transfer are synonymous. A "terms thesaurus" is typically used to group all synonyms under a "selected representative term".

Another important feature of the faceted classification scheme is a conceptual graph to measure closeness among terms in a facet. It is a weighted acyclic graph, in which the weight between two terms is assigned according to their closeness, as perceived by the user. So during a retrieval, if an exact match is not found, the terms close to the desired

term are tried.

Swanson and Samadzadeh [Swanson92] implemented the ideas of Prieto-Diaz and Freeman [Freeman87], and developed a prototype that can be used to catalog and retrieve software components. They have also implemented the "terms thesaurus" suggested by Prieto-Diaz and Freeman.

To form a new software system from the existing modules, there is usually a need for the composition of some of the modules. This in turn gives rise to the need for a formalism to specify the environment into which the modules can fit or to specify which modules can meaningfully fit a given environment. Algebraic structures can be used to describe the environment of a software module [Matsumoto87]. Algebraic structures consist of a set of sorts and the related operations. CLEAR is a language developed by Burstall and Goguen [Burstall80] for formal specification of software modules. CLEAR gives a formalism for the description of algebraic theories. It provides for building new theories from a combination of old theories, and for defining theory morphisms which are mappings between algebraic theories.

2.2.2.1 Modification and Combination of Modules. To build new software systems using composition technologies, it is necessary to be able to modify and compose software modules. In this subsection, the modification of software modules using parameterized programming and the combination of modules using Goguen's OBJ [Goguen89] are discussed.

Parameterized programming is a method by which the modification of existing parameterized modules can be done to make them suitable for use in a new software

system [Goguen89]. This is done by instantiating the parameters of the modules. Possible modifications include extending a module by adding to its functionality, renaming a module's external interface, and restricting the functionality of a module.

Consider the example (adapted from [Goguen89]) of a module LEXICO[X] that provides lists with lexicographic ordering. X can be instantiated to any preordered set, for example, the set of words[ID]. In such a way, LEXICO[ID] gives the lexicographic ordering of words and LEXICO[LEXICO[ID]] gives the lexicographic ordering of the sequence of phrases. Thus X is instantiated to the set of words.

Goguen's LIL (Library Interconnection Language) uses theories and views as the major semantic concepts [Goguen87]. Theories give semantic description of software components using axioms. Views describe bindings at interfaces. LIL allows the construction of new entities from the old ones in the following ways

- setting a constant,
- substituting one entity for a stub,
- sewing together two entities along a common interface,
- instantiating the parameters of a generic entity,
- enriching an existing entity with some new features,
- hiding some features of an existing entity,
- slicing an entity to eliminate unwanted functionality, or
- implementing one abstract entity using features provided by others.

The first four involve parameterized programming.

Goguen introduced the concepts of "vertical" and "horizontal" composition. In

vertical composition, higher-level abstractions are created from lower-level abstractions. Horizontal activities are to introduce views or to structure a specification, design, or a program, or to aggregate components into whole as needed before applying a transformation involving more than one component. Structuring activities are adding (combining) two or more entities, enriching (adding some new functionality to a entity), and deriving an entity from an existing functionality.

2.2.3 Generation Technology

In generation technologies, new software systems are generated by program generators from existing modules. There are two ways of doing this: by application generators or through program transformations.

A typical example of an application generator is the DRACO System [Neighbors89]. In this approach, when designing a software system for a particular problem area (domain), the needs and requirements of similar systems are analyzed so that, if needed, the software system can be modified to form those systems. Keeping the extracted requirements in view, the possibility of the implementation of the system using the domains already known to DRACO is pursued. Once a set of DRACO domains are developed, developing new software systems can be done for particular problem areas. This approach is very domain-specific and new software in the same domain can reuse only the existing software. DRACO can convert the internal representation back to the external syntax of the domain, optimize the internal representation, or implement the internal representation by software modules.

In the program transformation approach, an abstract program is first written using a very high-level language like SETL or GIST [Cheatham89]. Then this abstract program is refined into an actual program using two mechanisms: definition and transformation. Definition means providing a value for a procedure, type, or data object. Transformation means replacing some very high-level construct by a concrete construct that realizes the function.

CHAPTER III

DESIGN AND IMPLEMENTATION ISSUES

In a typical software development environment, each developer is aware only of the files associated with the current project and the previous projects on which the developer has worked. Usually there is no information at a central place on each file being developed as to which is accessible to whom, so that others can reuse the files. Therefore, developers end up reinventing the wheel rather than building on top of the previous experience.

Version control and configuration management systems are useful in managing the software development process by keeping track of the modifications made to the body and architecture of software. SCCS (source code control system) and RCS (revision control system) are software management tools that are available on typical UNIX installations (see Chapter II for more details on SCCS and RCS). This chapter discusses the design and implementation issues of a software tool VCI (version control interface). VCI provides a user-friendly interface to RCS and an environment for combining the concepts of software reuse with that of version management.

3.1 Design

To facilitate the integration of the concepts of reuse and version management, the

functionality of RCS was extended in the following way. In RCS and SCCS, once a delta is made, it is fixed and cannot be modified. If the software structure is to be reconfigured, it is desirable to allow for the insertion of deltas and the changing of the structure of the software module and its versions. These capabilities may be counterproductive or potentially disastrous in the hands of novice users. If the user is not careful with these capabilities, the whole system may become unreliable once the original structure is disturbed. Developers of version management tools such as SCCS have generally taken the stand that providing such capabilities are not desirable, because the indiscriminate insertion of deltas will lead to a situation where identifying the recently introduced deltas becomes difficult and debugging becomes virtually impossible. It is understandable that the availability of such capabilities is not desirable in the hands of a novice, but an experienced programmer need not be deprived of these capabilities. From a software repository point of view, this facility should increase the efficiency of storing files. As the library grows with time, it can be restructured to maximize the savings in storage space.

VCI is a tool for efficient storage of files that provides the capability to store files as versions of existing files. The concept of a delta, as used in SCCS and RCS, was extended to include any two files rather than versions of same file only. The tool compares a new file with the existing files in the repository. Only if there is no saving in space (achieved by storing the new file as a version of an existing file), is the new file stored completely. In the worst case, the file is stored completely and in the best case, considerable savings can be expected.

The space savings in storing files using VCI has a cost tradeoff in the form of the

time taken to access the files stored as versions of existing files. This time will in general be more than accessing those files stored in their entirety. This delay, due to accessing the files, should be tolerable considering the savings in space. A new file is stored as an RCS file in VCI if it is not stored as a version of another existing file to facilitate the storage of subsequent files as versions of this file. The overhead involved in storing a file as an RCS file is fixed and does not vary with the size of the file.

The tool also provides the capability of deleting the files that are already in the repository. This facility is again a two-edged sword. On one hand it can destroy the development history of the software project as explained before. On the other hand it is necessary to be able to remove the files no longer required in the repository. To minimize the danger of the important files being deleted, the deletion of files initiated by the users is not effected until the library administrator approves each deletion.

One advantage of VCI over typical configuration management systems is that files from different projects can easily be composed into a meta project. Suppose a new project can be built by coding ten programs and further suppose that four of them are already available to be used as is or can easily be adapted from different existing systems, then the project can in general be built in less time by composing these programs with the other required programs. This scenario can work only if the following conditions are satisfied.

- The files can be identified as being available,
- The files are accessible, and
- The files are understandable.

VCI catalogs the files (programs) using the faceted classification and stores the files at a central library. Retrieval methods provided by VCI can be used to verify if a file (or program) with the given functionality is available. If available, the file can be checked out into the current directory of the user.

The word library used in this thesis connotes a collection of files. The collection can be someone's personal collection or that of an organization. The tool can be used to create and maintain any such collection.

The above capabilities may appear to be undermining the fundamental principles of version management. After a prudent analysis it will be observed that the above facilities are provided to extend the typical version management capabilities. If VCI has to be used strictly as a version management tool, all requests for moving files and the deletion of files can be ignored by the library administrator. The tool can be used to manage some of the developmental branches using the regular version management methods, and some other branches using the methods described above. The library can be visualized as a set of branches, with each branch being an individual project, where the current projects are managed using regular version management techniques.

In the very least, VCI is designed to function as a version management tool with an efficient method of organizing the project (that is, a group of files). Of course, the tool works as an interface to a software repository with a number of files catalogued and stored efficiently.

3.2 Implementation

To provide the above functionality, that is, extending the functionality of RCS and

providing an efficient software repository, seven options are provided. They are "Libin", "Libout", "Link", "Move", "Delete", "Options", and "Stats". The software repository was designed to contain a group of RCS files. This is transparent to the user. The user is provided with the view that the repository is collection of software components classified by their functionality. To maintain this transparency, a file containing some information about the software such as name, the revision number in the corresponding RCS file, and the number of branches is maintained by the tool.

When a user wants to check-in a file into the library, the user is presented with a list of files from which the user can choose the file to which the user wants to check-in the new file as a version. Since the selection of the file, to which the new file is made a version, seems to be important to maximize the savings in space, the user is provided with the option of allowing the system to choose the file to which the new file can be checked-in as a version. This alternative option has a disadvantage in that it is time consuming for the system to match a given file with all the files in the repository and come up with the best match. Once the name of the file, to which a new file is to be made a version, is identified, the corresponding index is looked up in the information file. Using the index, the information of the file (such as its revision number, RCS filename, and logical name) is retrieved. From this information, the name of the corresponding RCS file and the revision number of new branch is deduced. Using this information, the new file is checked in as a version.

To capture any additional functionality of a file that has been checked in, a user can link the file to additional keywords (words capturing each of the additional functionalities)

using the link facility. The link facility links a keyword to a file and also records the difficulty in adapting the file to perform the function stipulated by the keyword.

To check out a file from the library, a user is provided with a list of keywords (such as sorting, searching, list handling, and compressing) from which the user has to select one that closely describes the expected functionality of the required software. Using the graphical display of the tree of the library structure, files performing similar functions are reviewed (since the files performing "similar" functions are expected to be "together" in the tree) and the exact file that needs to be checked out is located. Once the file is located, its RCS file name, version number, etc., are looked up from the information file. Using this information, the corresponding file is checked out.

Once a file is checked in, if the user wants to unlink a file from its current parent (or the file to which it is a version) and link it to a new parent (that is, some other file), then the user can use the "Move" option. The move operation is performed (actually effected) only after the system administrator allows the same. The system administrator can initiate this action at any time and the corresponding changes are accomplished. The changes in the physical files are done off-line and the system is not affected during its operation.

To maintain the repository, the administrator is provided with the "Stats" option. With this option the administrator can plot the graphs for the space saved and the frequency of check-out. These graphs can be used to judge the performance of the system. The files which are not frequently checked out can be removed from the library.

The characteristics of the code to implement VCI was as follows. The screen management code was around 4300 lines, and the code implementing various VCI

functions was approximately 4200 lines. The documentation of the VCI code was about 1600 lines. Header files and data files contributed around 1600 lines.

3.3 User Interface

The user interface of VCI is described below. The initial screen accepts the password from the user (see Figure 6). The user can use the backspace keys, the left key, the right key, and the delete key of the keyboard while typing the password. Once the "Return" key is pressed, the password typed is checked for validity against the password list. If the password is wrong (i.e., not validated), a warning message is popped up on the screen informing the user that a wrong password has been typed. The user is allowed to retype the password. The second screen consisting of several buttons and a drawing area widget is displayed (see Figure 7), once the password is verified. Each button has a grey background and an appropriate title as its label, as described below.

When the "Libin" button is pushed, the user is presented with a file selection dialog box. The file selection box allows the user to select a file to be checked into the library. The dialog consists of two text widgets, two list widgets, and four pushbuttons. The first text widget is the filter widget. The pattern in this text widget is separated into two parts when the "Return" key is pressed or the filter Pushbutton is pushed. The first part is for the directory and the second part is for the pattern. For example, if the text in the filter widget is `/m/nadella/*.c`, the directory part is `/m/nadella` and the pattern part is `*.c`.

All the directories in the directory part are displayed in the directory list. All files matching the pattern part are displayed in the file list. In the above example, all files

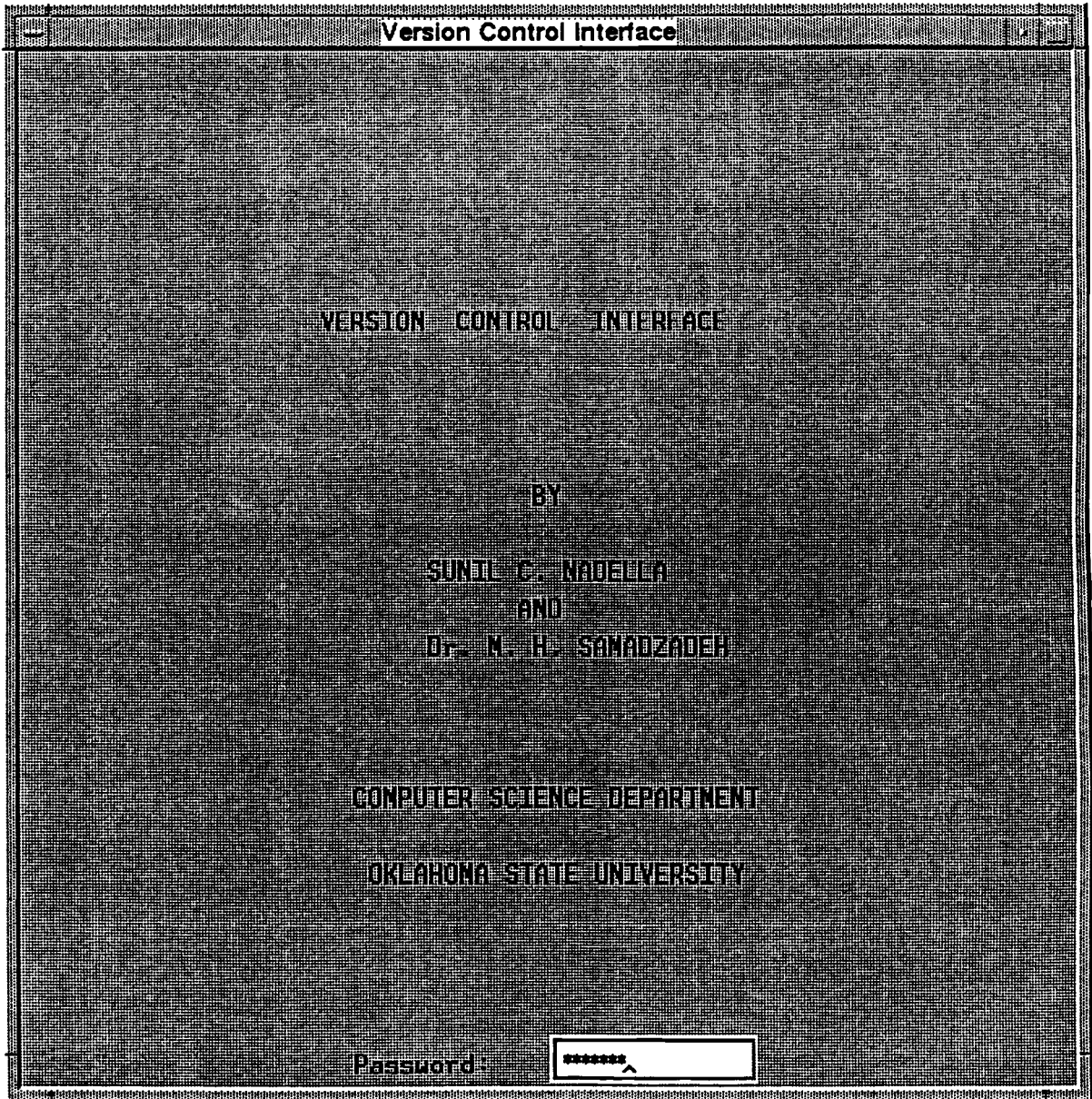


Figure 6. Initial screen

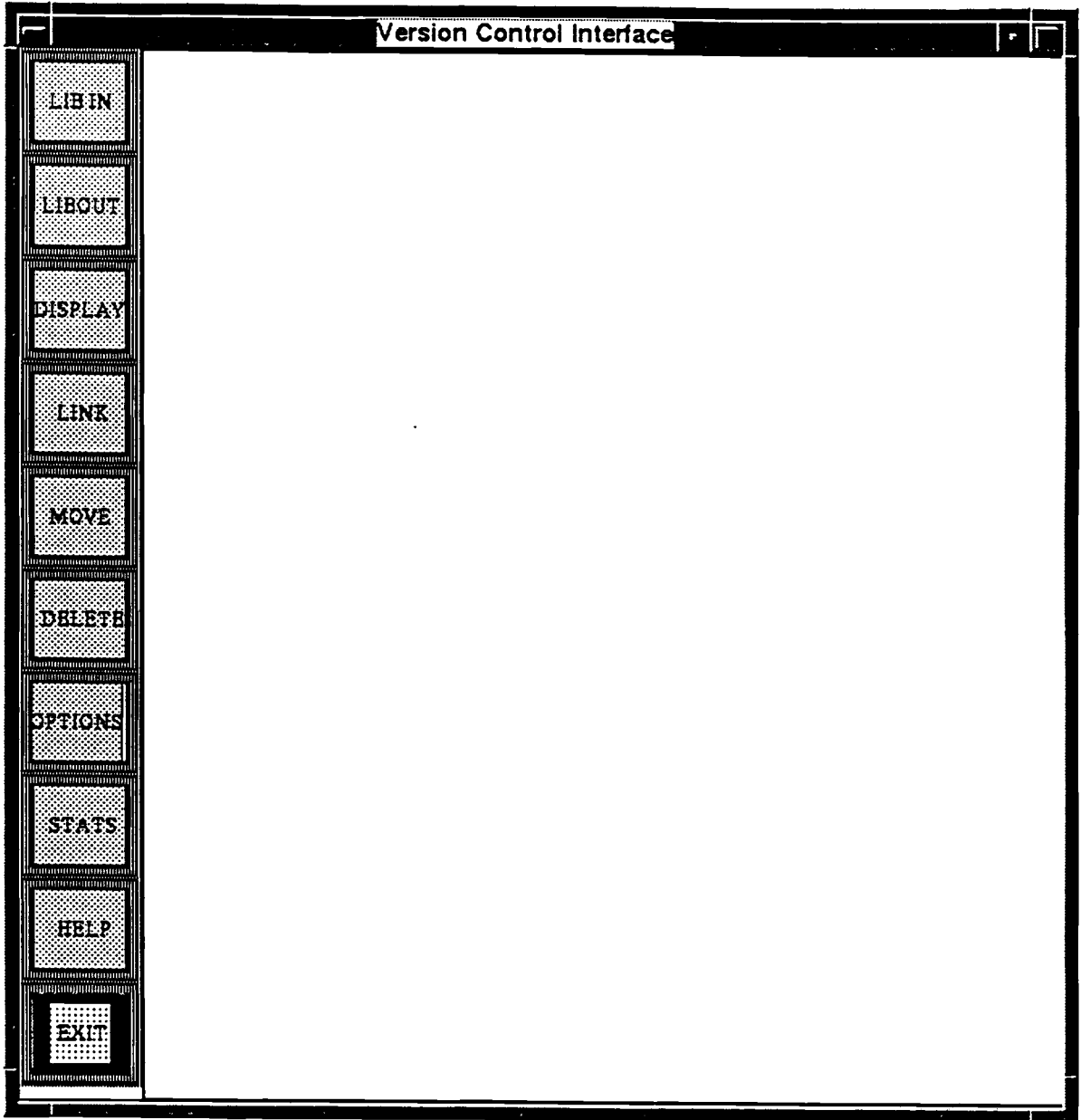


Figure 7. Screen with drawn buttons

ending with ".c" are displayed in the file list. In the directory list, double-clicking on any directory displayed changes the current directory to that directory. Double-clicking on a file in the file list completes the selection of the file, and the file selection dialog box is dismissed. A single-click on any file name puts its full path in the selection text widget. If the "OK" button is pushed, the file in the selection text widget with the path is marked as the chosen file. The "Cancel" button is provided to abort the check-in process without selecting any file. If a valid filename is not provided in the selection widget and "OK" is pressed, the file selection dialog is not dismissed.

Once a file is selected for checking in, the user is presented with a description dialog (see Figure 8). It consists of four text widgets, one scrollable text widget, and two pushbuttons. The text widget can be used to provide the description of the file to be checked in as described in Section 2.2.2 in Chapter II. The "Cancel" pushbutton can be used to abort the check-in process. The "OK" pushbutton when pushed, records the description and presents the user with a choice of either allowing the system to place the file at an appropriate position in the library, or selecting the position personally. This dialog has two radiobuttons to allow the user to make a selection in addition to the "OK" and "Cancel" pushbuttons. The "OK" button records the choice made by the user, and the corresponding action is taken. If "Position selected by system" is selected, no more dialogs are presented and the check-in process is complete. If "Position selected by user" is selected, the user is presented with a List dialog (see Figure 9). The List dialog has one text widget, one scrollable list widget, and three pushbuttons. The text widget is used to select a file in the library to which the current file is attached.

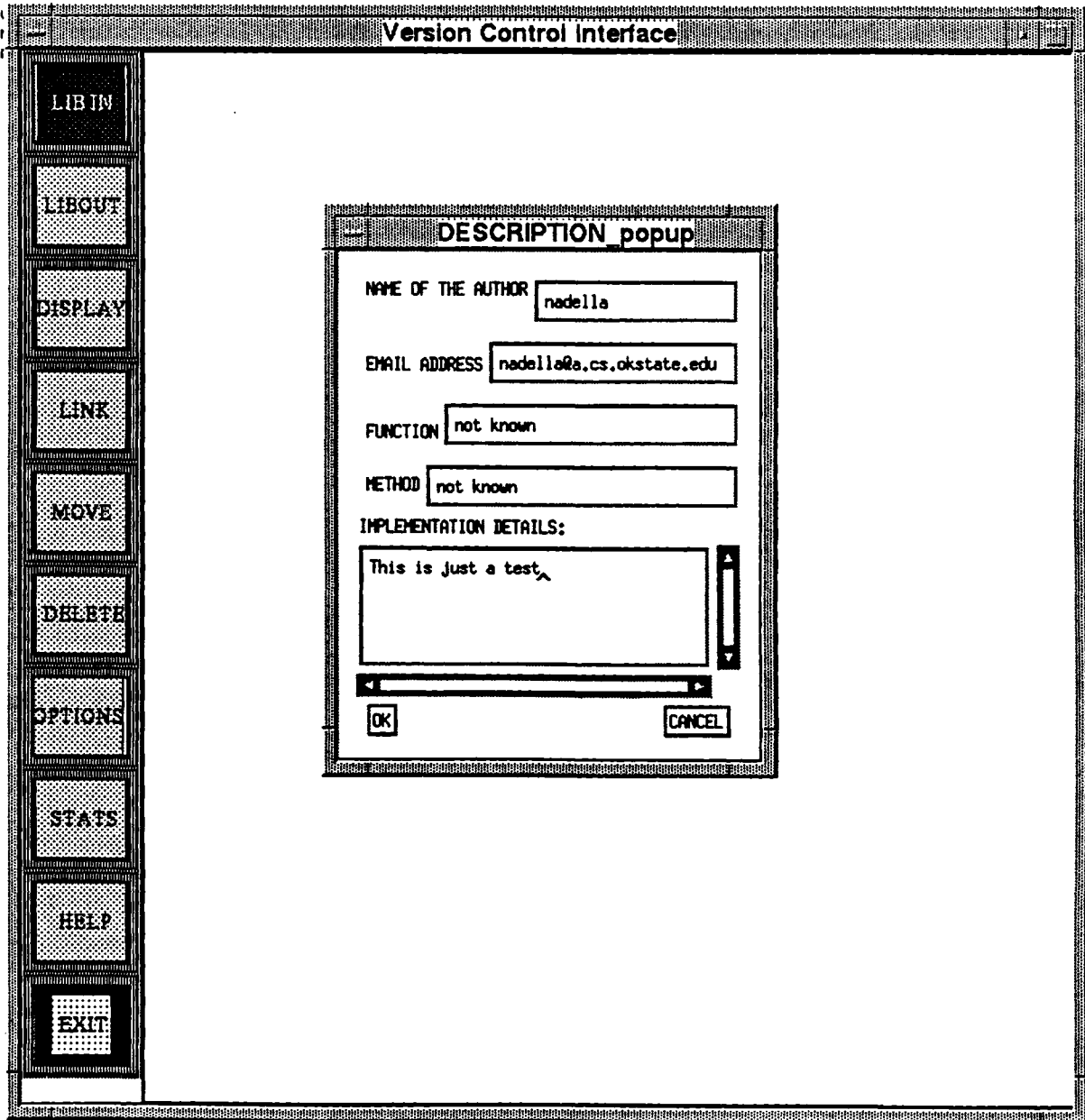


Figure 8. Description dialog

The meaning of Link and Thesaurus is explained in detail in Section 2.2.2 of Chapter II. If a user types in a Link word (i.e., an alternate words capturing additional functionalities of files) in the text widget and presses the "Enter" key, the user is presented with a list of files associated with the link word. The user can select a file from this list by seeing the description of the file by double-clicking on a file name. The description is presented in a scrollable text widget in a dialog box. Pressing the "OK" button in this dialog box puts the file name in the "Select File" text widget. The functionality of being able to view the description of a file by double-clicking on its name is also provided in the List box. If the Thesaurus button is pushed, the equivalent words in the thesaurus of the library are presented as a list. Double-clicking on a word posts the word in the "Select File" text widget, and dismisses the thesaurus list. If the word is not found in the thesaurus of the library, an error message is displayed in an error dialog box.

Once the "OK" button is pushed, the text in the "Select file" text widget is recorded as the selected file, and the check-in process is started. If any of the two chosen files (one in the file selection dialog and the other in the List dialog) are invalid or not found, the corresponding error-messages are given and the check-in process is aborted.

If the "Libout" drawn button is pushed, the user is presented with a choice of selecting the file to be checked out by name from a list, or by searching for a word (or pattern) in the description of files. This choice is to be made by pushing one of the two pushbuttons "File selected by name" or "File selected by pattern matching" in the dialog. Since selecting files by name is the same as selecting files with a certain functionality encountered in the last stage of the check-in process, the same dialog as in check-in is

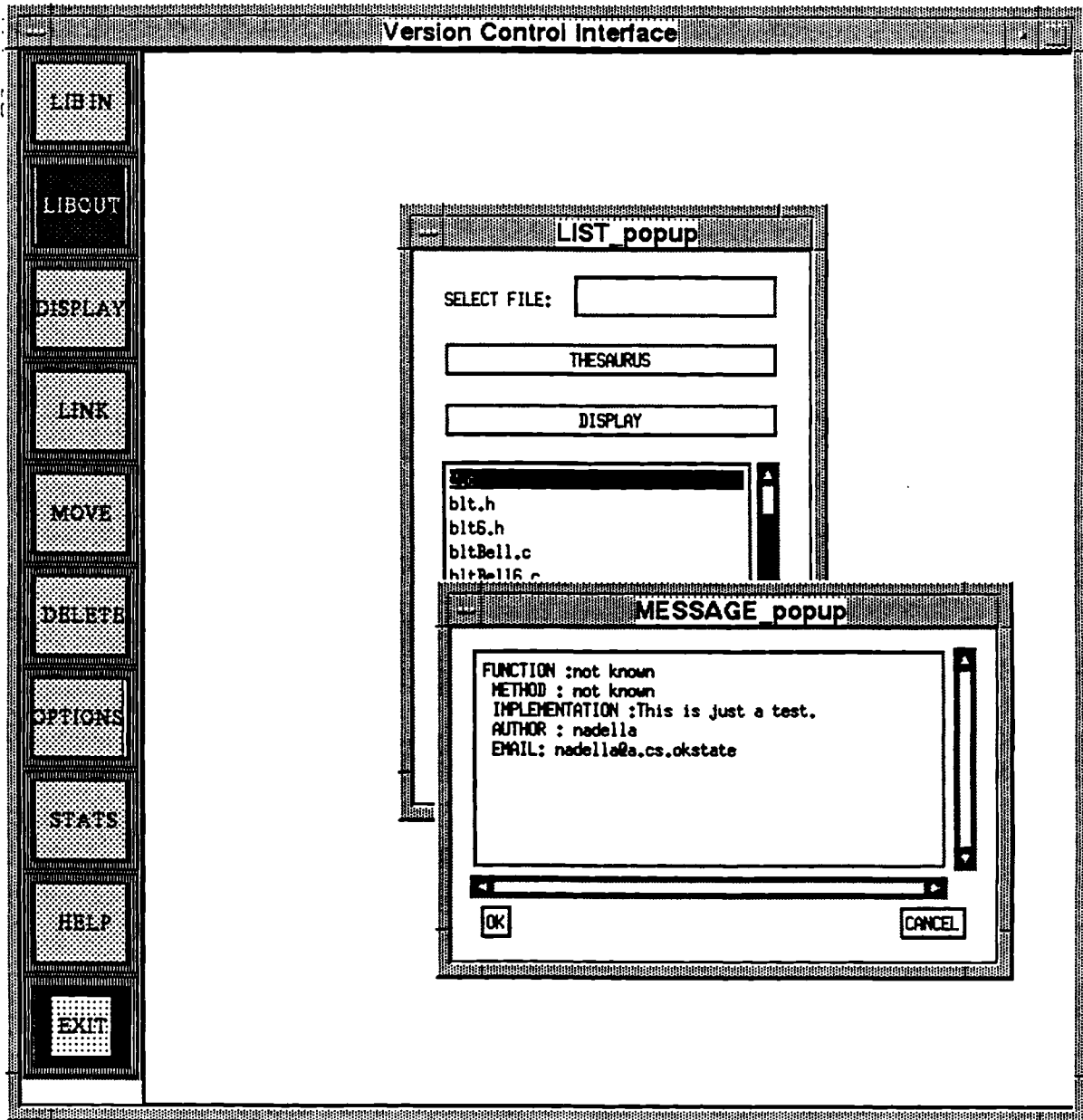


Figure 9. List dialog

presented. If selection by pattern matching is selected, a dialog with a text widget is presented to the user to obtain the pattern to be searched in the description of the files. Once the "OK" button is pushed, the search process is initiated, and all files with the pattern in the description are presented as a list in a dialog (see Figure 10). Double-clicking on a filename gives the description of the file (as in other lists described), and pushing the "OK" button puts the filename in the "Select file" text widget. When the "OK" button is pushed, the filename in the "Select file" text widget is marked as the file to be checked-out and the corresponding check-out operations are initiated. If the file name is invalid, an error message pops up indicating the same.

If the "Display" drawn button is pushed, the user is presented with a text widget in a dialog shell prompting the user to enter the filename from which the library structure is to be displayed. If an invalid name or no name is entered and the "OK" button is pushed, the structure is displayed from the head or the starting node. The structure is displayed as a tree with files represented as square boxes in the drawing area widget. There are two modes of showing the structure: Zoom-in and Zoom-out. The zoom-in mode has one file as the current file. This is the file from which the display was initiated. The children of this file (i.e., the files connected to the current file) are displayed from top to bottom to the right of the current file. Due to space constraints, a maximum of three children are displayed. If there are more than three children, they can be seen by clicking on the two arrowheads at the right edge of the current file box, as needed. The parent of the current child (i.e., the file to which the current file is connected) is displayed to the left of the current file. Clicking on any file box makes it the current file. In this

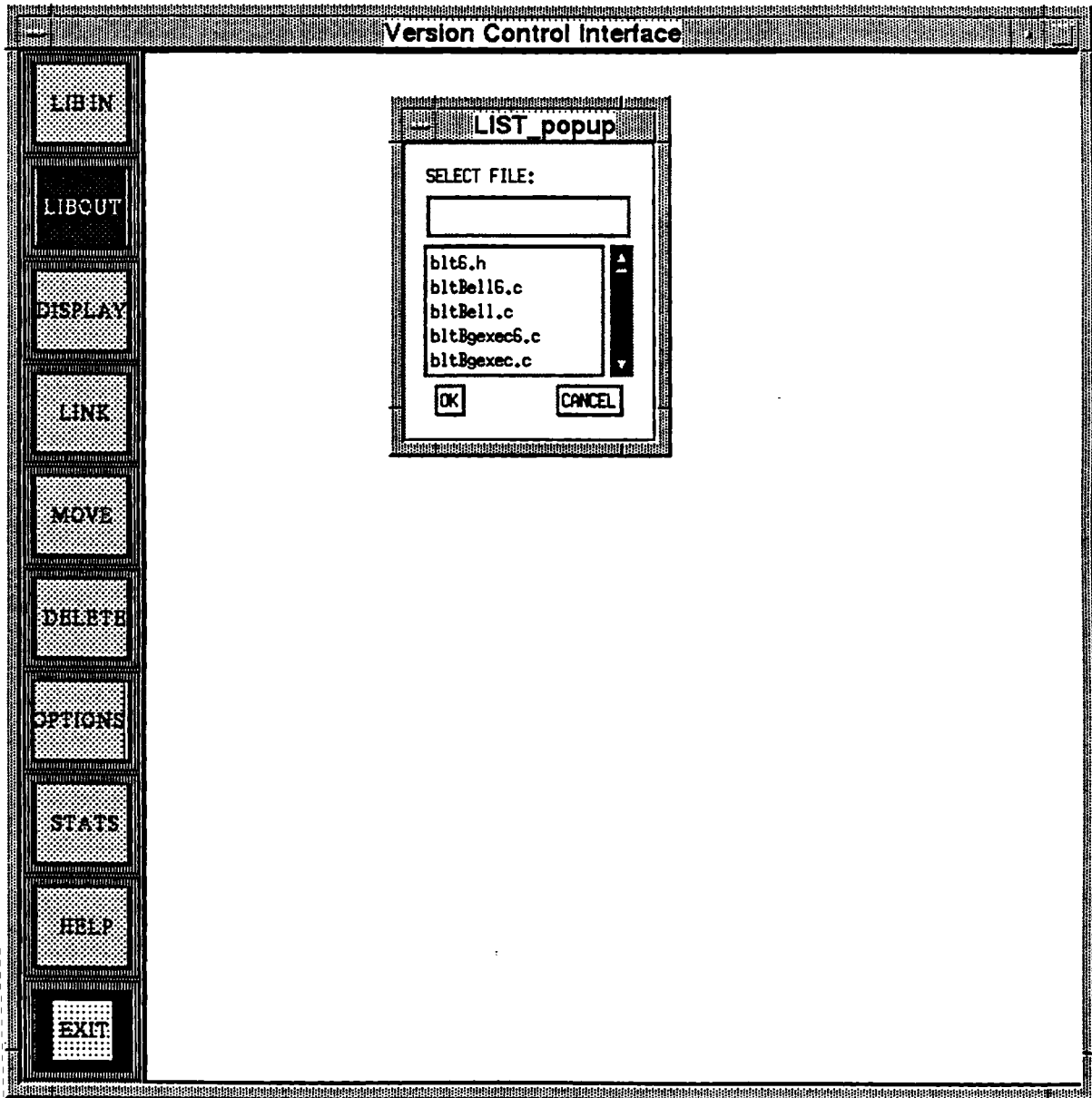


Figure 10. Pattern search dialog

way navigation through the library structure can be achieved.

In the zoom-out mode the current file in zoom-in mode is first displayed as a rectangle at the right side of the display. Then its parent and ancestors are displayed to its left. The number of nodes below this node including itself, and the height (distance from the leaf) are displayed in the rectangle. Its siblings are displayed above and below the node. At most five levels are displayed horizontally. If there are more than five levels between the head and the current file, the number of levels is displayed in a triangle to the left of the last node displayed. Subsequent zoom-outs will compress the first five levels, and the next five levels are displayed, and so on. At most four siblings are displayed at each node. If there are more than four siblings, a rectangle is displayed at the bottom. This rectangle displays the number of nodes in the unseen siblings and the maximum height of the hidden siblings. By pushing the zoom-in or zoom-out pushbutton as needed, the mode of the display can be changed. The exit button can be used to dismiss the display widget.

When the "Link" drawn button is pushed, a dialog with three text widgets is presented, prompting the user to enter the name of the link, the name of the file to which the link is to be made, and the distance. The concept of a link is explained in Subsection 2.2.2 of Chapter II. If a valid filename is not given or a previous link with the same name exists to the same filename, an error message is given in an error dialog.

When the "Move" drawn button is pushed, the user is prompted to enter the name of the file to be moved and the name of the file to which the above file is to be attached. When the "Delete" drawn button is pushed, the user is prompted to give the name of the

file to be deleted. The "OK" button initiates the "Delete" operation and the "Cancel" button aborts the "Delete" operation.

To change the description of files checked-in or to contribute to the thesaurus, the "Option" drawn button can be pushed. Either "Change description" or "Thesaurus" can be chosen by clicking on the corresponding label and pushing the "OK" pushbutton in the dialog that pops up. In the "Change description" option, a dialog with a text widget pops up requesting the name of the file whose description is to be changed. When the "OK" pushbutton in this dialog is pushed, a description dialog similar to the description dialog in the check-in process pops up with the current description of the file in the various fields. The description in the various fields can be changed and the "OK" button can be pushed to end the operation. In the "Thesaurus" option, a dialog pops up requesting the user to give the word to be included in the thesaurus. Once the "OK" button is pushed, a dialog with one text widget, an "Add" and a "Delete" pushbutton, and a list widget, in addition to the "OK" and "Cancel" pushbuttons, pops up (see Figure 11). The equivalent words can be typed in the text widget. On pushing the "Add" pushbutton, the word is added to the list widget. Similarly, pushing the "Delete" pushbutton deletes the word from the list widget. Once the "OK" pushbutton is pushed, the words in the list widget are recorded as the equivalent words.

When the "Stats" drawn button is pushed, the user is presented with a dialog box with the names of the various graphs that can be viewed as radiobuttons. Choosing a graph by clicking on the corresponding radiobutton and then pushing the "OK" button causes the graph to be plotted using BLT [McLennan93]. There are two pushbuttons in

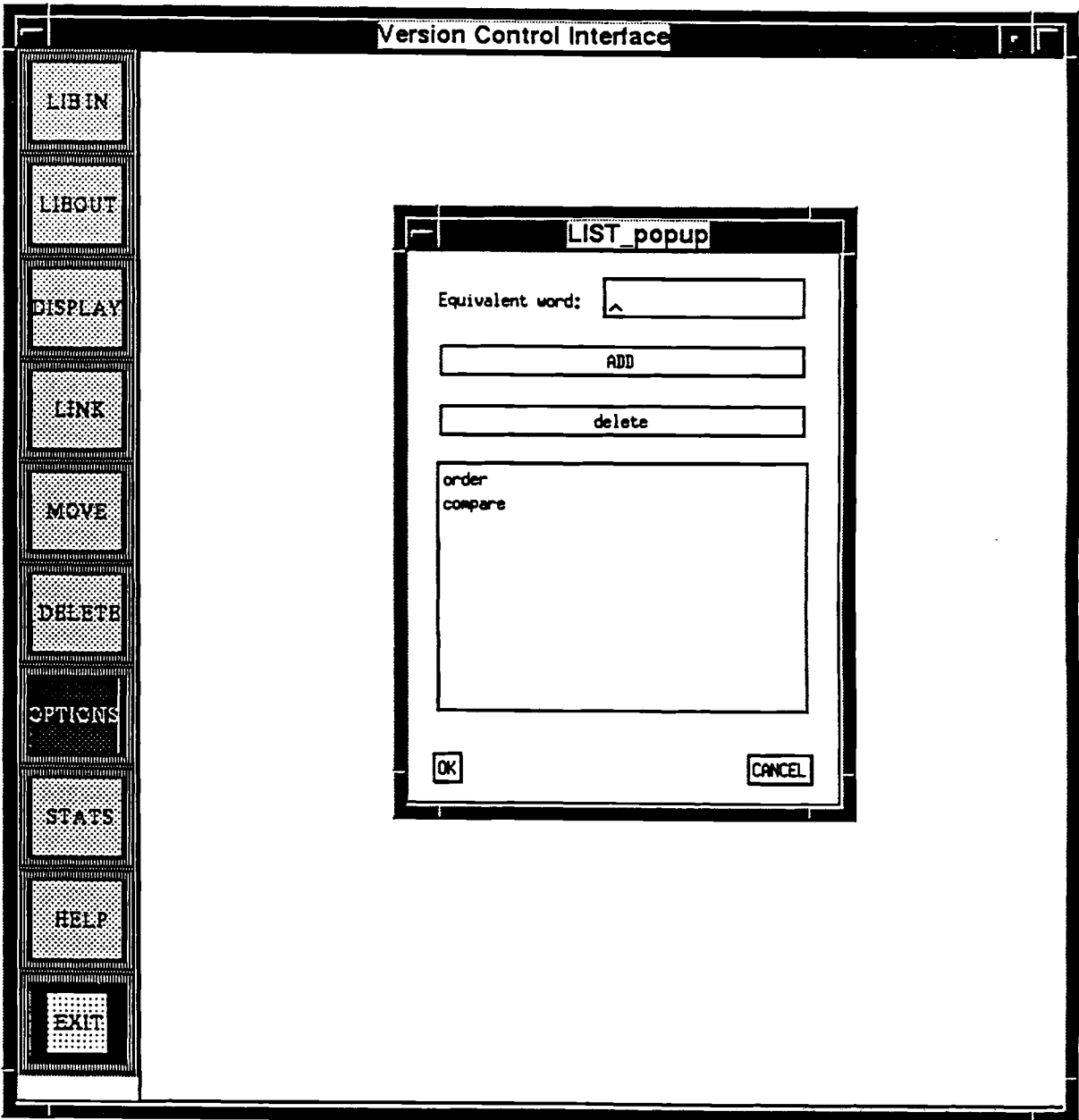


Figure 11. Thesaurus dialog

each graph: "Print" and "Quit". When pushed, the "Print" button writes the graph into a postscript file that can be printed using a postscript printer. When the "Quit" button is pushed, the graph is dismissed.

When the "Exit" button is pushed the user is presented with a warning dialog enquiring if the user really wants to quit. If the "OK" button is pushed, the application is closed.

3.4 Implementation Platform and Environment

The Sequent S/81 System is a mainframe-class multiprocessor system [Sequent90]. S/81 runs the DYNIX/ptx operating system which is Sequent's implementation of the UNIX operating system. The availability of multiple processors improves the performance of sequential applications by improving system throughput. Programs can be automatically or manually made to use multiple processors simultaneously, thus increasing overall execution speeds.

The Version Control Interface (VCI) was implemented in the X Windowing system using the Motif toolkit on the Sequent S/81 system. The X Windowing system was developed at MIT in 1984. Applications built in the X environment are portable and can run on any machine supporting X, which is based on the client-server model. The display servers provide display capabilities and accept input from the users. The clients are the application programs. X is not device dependent because of the existence of its network protocol.

Motif was designed by the Open Software Foundation [Heller91]. Motif is a set of guidelines that specifies the look and feel of user interfaces for graphical systems. The

Motif toolkit is based on the X Windowing system. Using the toolkit, Motif compliant applications can be produced easily.

CHAPTER IV

SAMPLE SESSION AND OBSERVATIONS

4.1 Sample Session with VCI

A sample session with VCI is provided in this chapter to illustrate the functionality of the tool. For the sample session, an attempt is made to check in a file for a program that draws a tree in a dialog, and check out a file for a program that sorts an array of numbers. The initial screen after giving the password is shown in Figure 12 (this figure is the same as Figure 7, the repetition is for convenient reference). The "Libin" button is pushed to start the check-in process. The name of the file (draw.c) is given in the file selection dialog. The description of the file is given in the description dialog (see Figure 13). The "Position selected by user" method is chosen in the next dialog. In the list dialog, the word "draw" is typed in the text widget to see if any file with a similar name is available in the library. A list dialog pops up with the information that draw is connected to file Figure.c (see Figure 14). The tree of the library structure is then viewed from Figure.c by pushing the "Display" pushbutton (the zoom-out mode of the display is shown in Figure 15). The file Figure.c is chosen as the parent of the file draw.c by pushing the "OK" button. The check-in process is completed.

To initiate the check-out procedure, the "Libout" button is pushed. The user is presented with a dialog box displaying two retrieval methods. First the "Selection by file

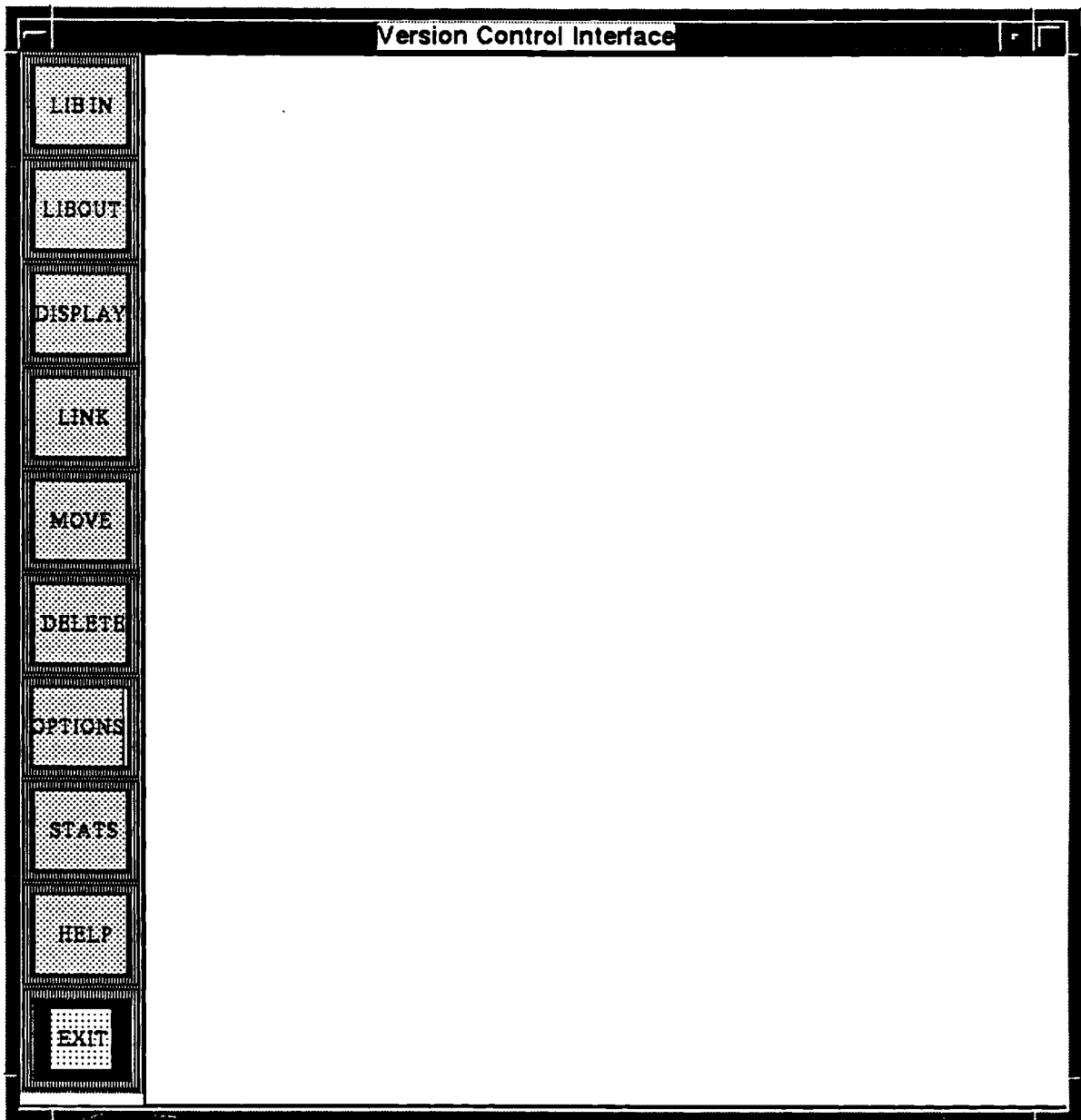


Figure 12. Screen with drawn buttons

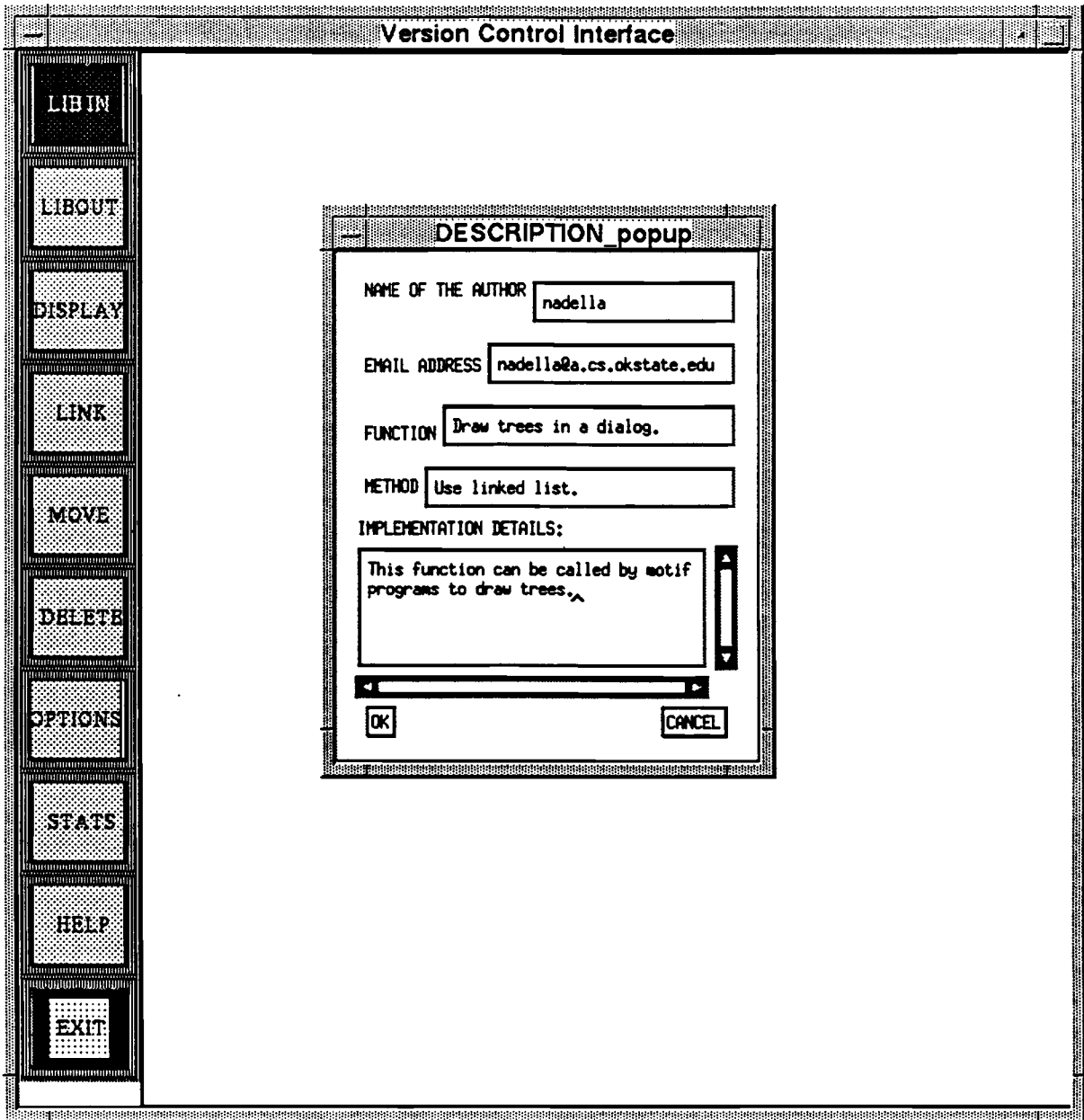


Figure 13. Description dialog of draw.c

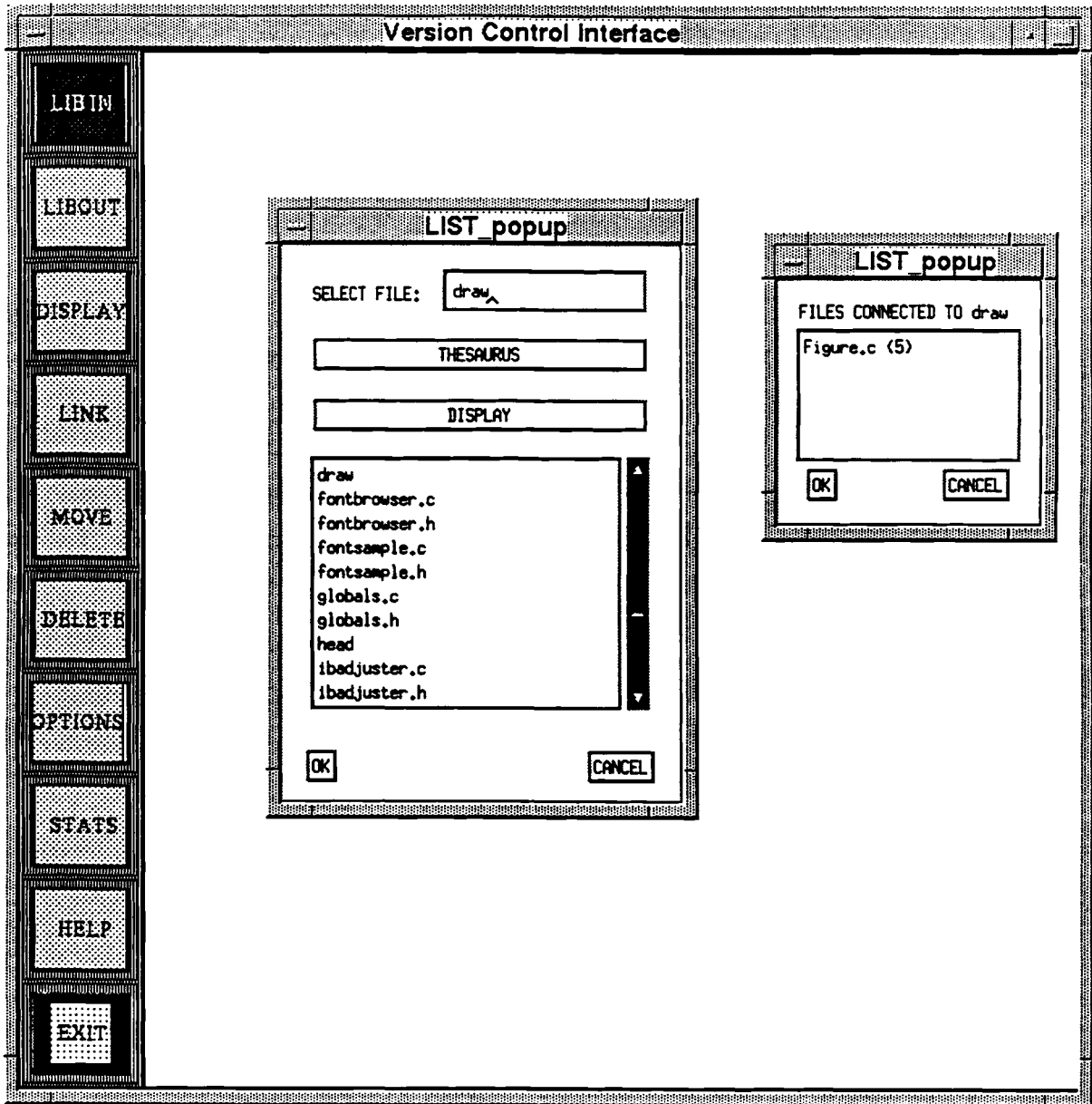


Figure 14. List dialog of draw.c

name" option is attempted by selecting it and pushing the "OK" button. A dialog with a list of files and logical links, shown in a list widget, is presented to the user. To check if any files are named as "sort", or to see if any files are linked to "sort", the word sort is typed in the text widget. The tool responds by showing the files associated with the link "sort". The description of each file can be viewed by double-clicking on its name in the list. In this sample session, this action isolates the file bubblesort.c as the required file. The same list of files can be obtained if the "Pattern search" retrieval method is selected and the pattern given is the word "sort". The library structure can then be viewed by pressing the "Display" button. The file from which the library structure is viewed is bubblesort.c. A snapshot of the display screen in zoom-in mode is shown in Figure 16. The zoom-out mode, viewed by pushing the zoom-out pushbutton, is captured in Figure 17. The statistics shown by the tool can be viewed by depressing the "Stats" button, and then choosing the space utilization graph and the check-out frequency graph one after the other. A sample graph for space utilization is shown in Figure 18, and a sample graph for the percentage space utilization is shown in Figure 19.

4.2 Observations and Limitations

The tool (VCI) was made available to the MS and PhD students of the Computer Science Department at OSU. Several of the graduate students tried VCI and offered a number of suggestions. Their suggestions were carefully considered and changes were made to the tool.

With the experience obtained in maintaining the library while VCI was prototypically

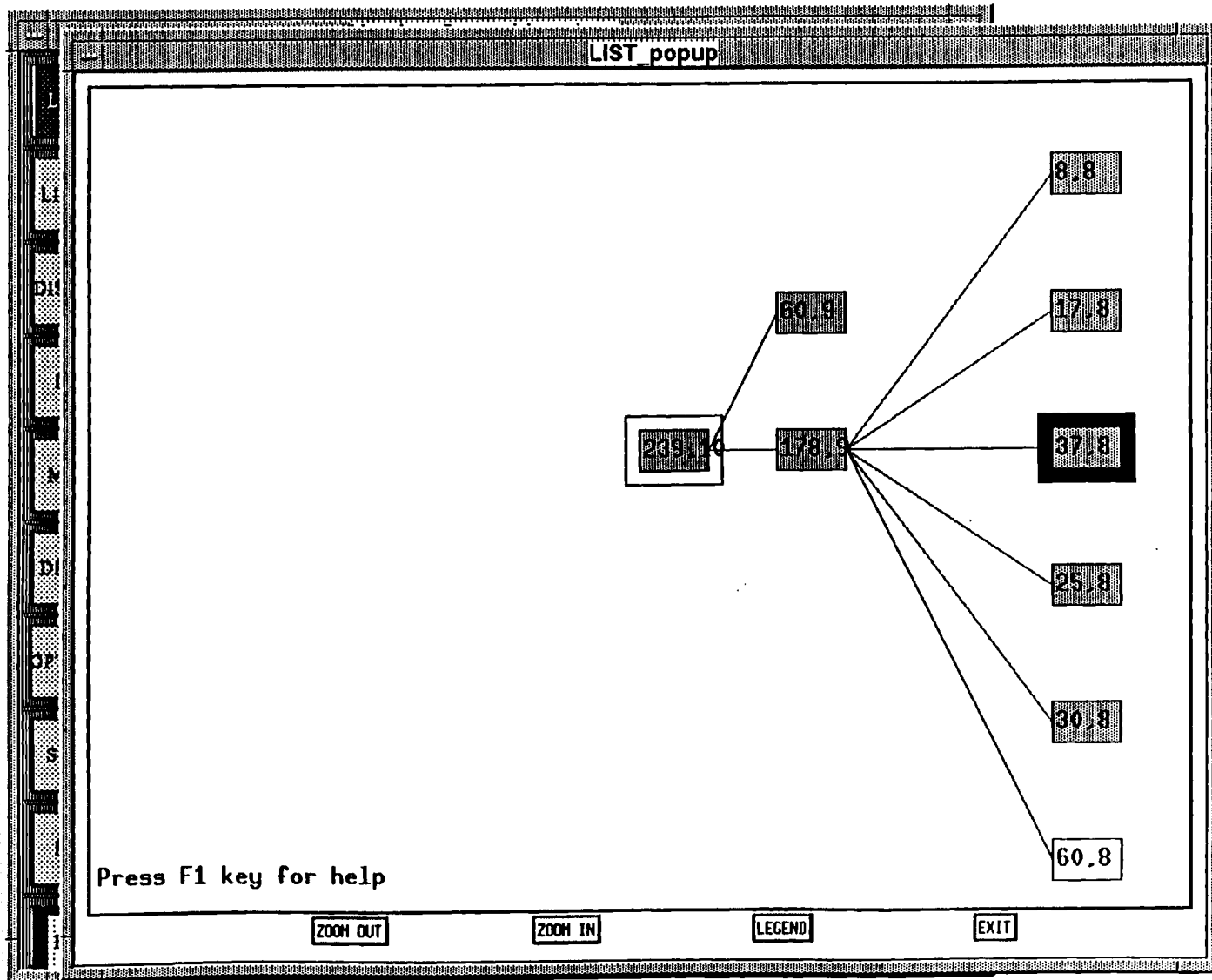


Figure 15. Zoom-out mode of the display from the Figure.c file

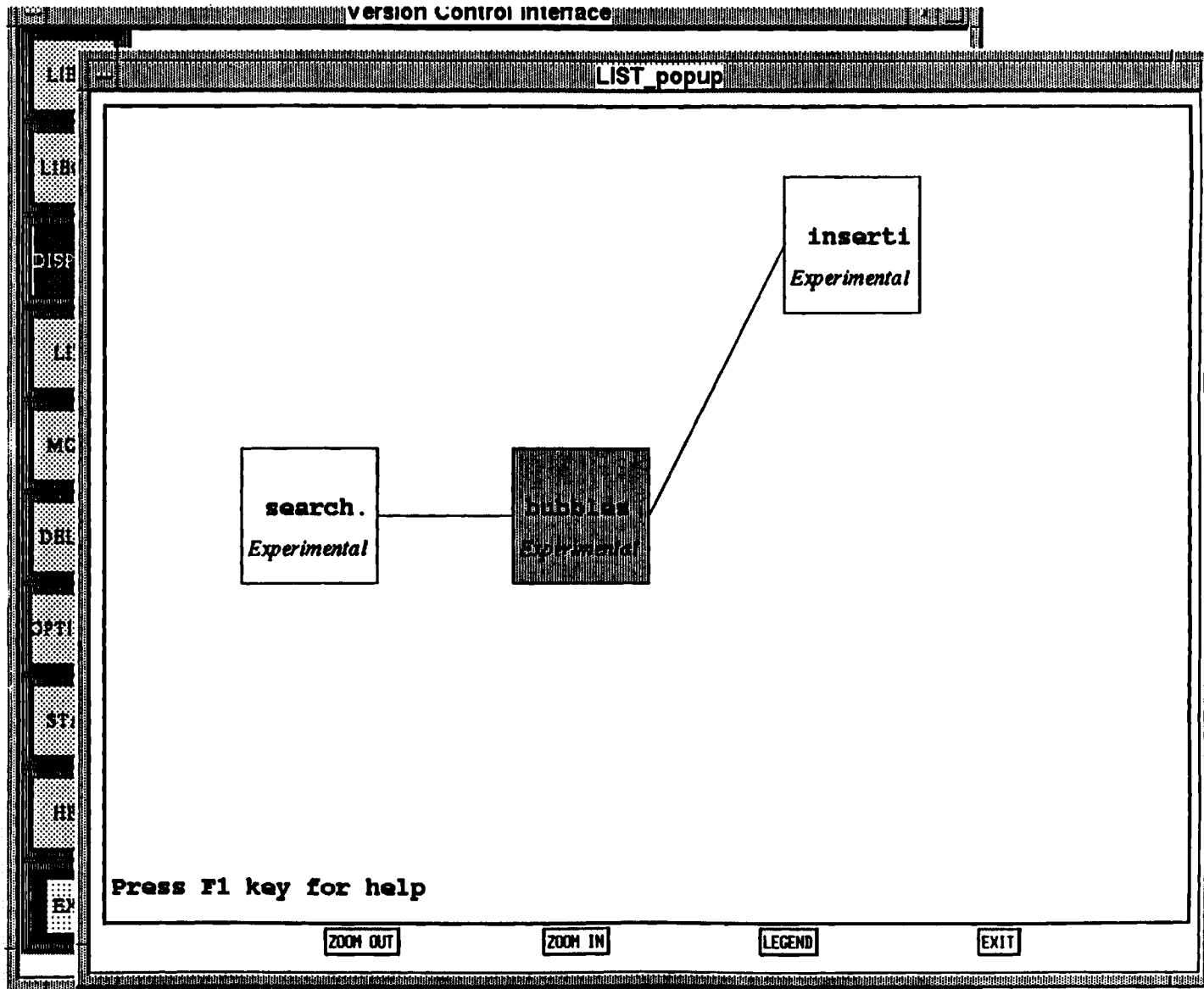


Figure 16. Zoom-in mode of the display from the bubblesort.c file

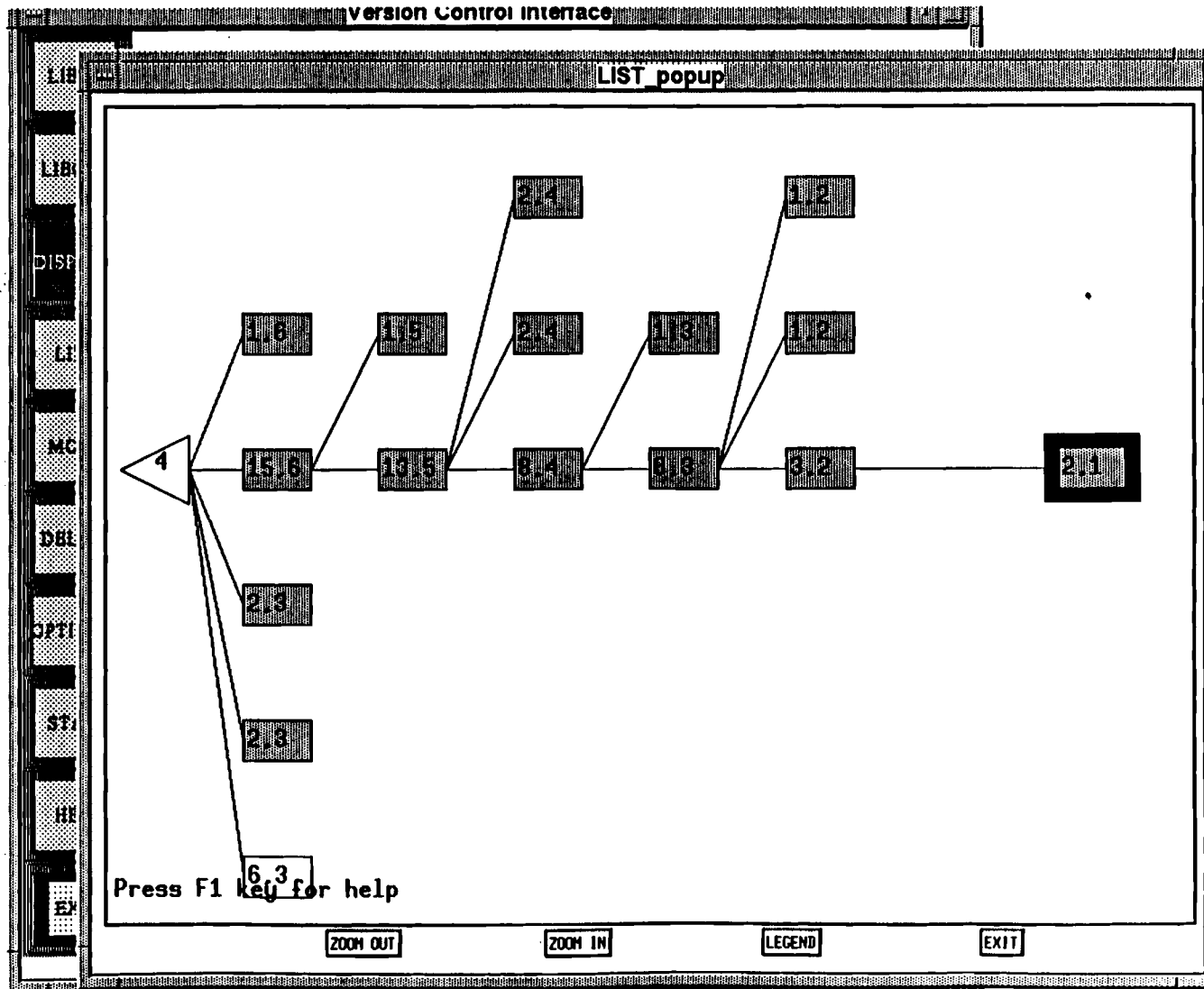


Figure 17. Zoom-out mode of the display from the bubblesort.c file

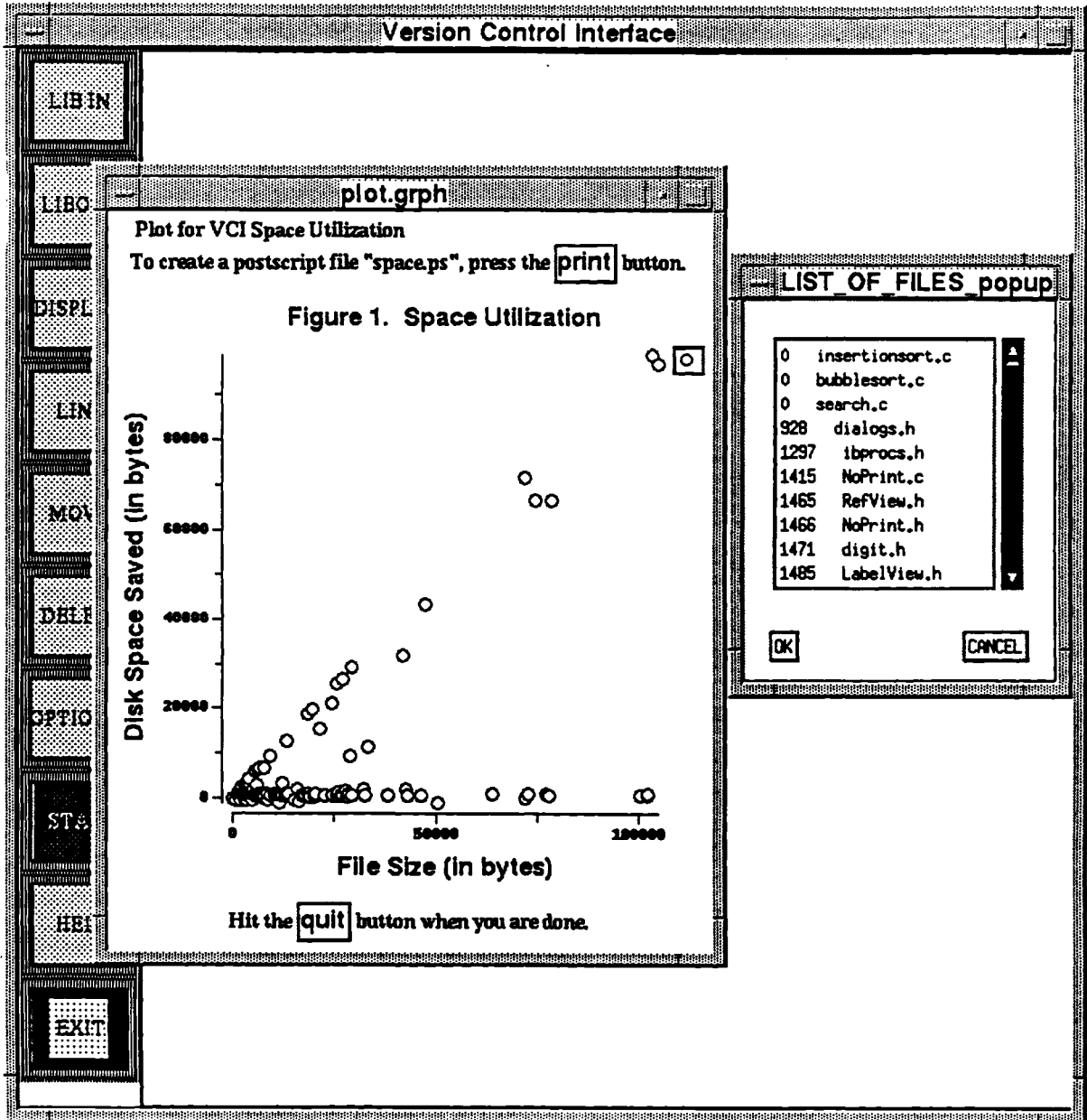


Figure 18. Space utilization graph

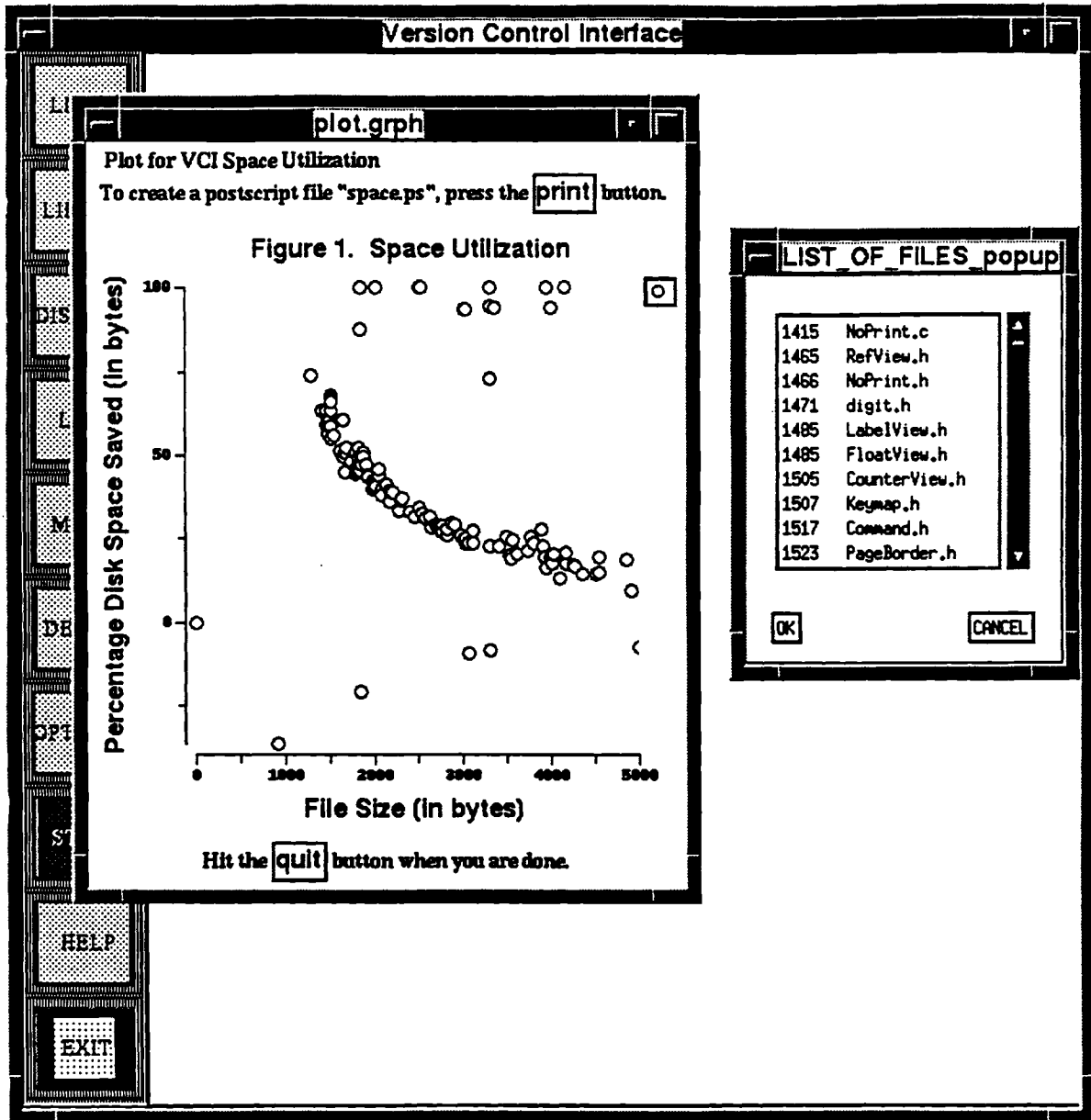


Figure 19. Percentage space utilization graph

tested, the following observations can be made. The main problem associated with the tool was the difficulty faced in giving the description of the files to be checked in. The users generally preferred to ignore giving the description as it is considered tedious to come up with a description. This was understandable because the users were not long-term users, rather they were just testing the various functionalities of VCI. Hence, a short-cut was designed. Users who preferred to give a shorter description were allowed to provide only the name, the e-mail address, and the function portion of the description.

The other difficulty faced was in providing an accurate distance parameter in making a link. This parameter is dependant on the perception of the user and, with no guidelines available, the task of furnishing the distance becomes very difficult. A temporary solution to this problem is to use the difficulty of adapting a file (in terms of the number of hours that may have to be spent (a possible approach would be the effort or time measures of software science [Halstead77])) to perform the function connoted by the link. Validation of this proposed solution and its possible refinement to a more plausible solution can only be made after gaining experience in actually using the tool extensively, and hence is deferred to future work.

The other drawbacks of the tool are as follows. Files with control characters cannot be checked in, since RCS does not support files with control characters. Hence, executable files cannot be checked in. The integrity of the VCI system files is very important for the correct functioning of the tool. The structure and the description of these files are presented in Appendix C. If the main.dat file is lost, the whole library system may be lost. The files can be recovered from the RCS files but mapping them to their

respective names is difficult. Even though the user interface of VCI is friendly, with appropriate online help provided, users may still be reluctant to use this system. The prevailing mentality being, if one can get along without using this system, why use this system at all. It can be argued that creating a system such as VCI is one half of the story. The other half is educating the potential users to use the system.

CHAPTER V

SUMMARY, CONCLUSIONS, AND FUTURE WORK

5.1 Summary

An user friendly interface to RCS called Version Control Interface (VCI) was built using the Motif toolkit. VCI integrates the concept of software reuse with version management techniques. The use of VCI in building and maintaining a software repository was demonstrated.

5.2 Conclusions

VCI can be used as an interface to a software repository. Files are stored efficiently by VCI by storing them as versions of similar existing files in the repository. When a user needs a file, the user can check if a similar file is present in the repository and use it. In this way, VCI is a stepping stone to a future where software is built more from existing parts than from scratch.

5.3 Future Work

The concept of distance, as explained in Section 4.2 of Chapter IV, must be investigated thoroughly to provide a proper formalism to specify it. The method of providing a description for a file should be further tuned to capture the accurate description of the modules. The tool can be modified to compress and encrypt the RCS

files. Compression should give additional savings in space. With encryption the files in the library are more secure. The tool can be extended by providing operations to compose software from existing software.

REFERENCES

- [Babich86] Wayne A. Babich, *Software Configuration Management*, Addison-Wesley Publishing Company, MA, 1986.
- [Biggerstaff89] Ted J. Biggerstaff and Charles Richter, "Reusability Framework, Assessment and Directions", in *Software Reusability, Vol. I*, Ted J. Biggerstaff and Alan J. Perlis, Eds., pp. 1-18, Addison-Wesley Publishing Company, NY, 1989.
- [Burstall80] Rod Burstall and Joseph Goguen, "The Semantics of CLEAR, a Specification Language", *Proceedings of the 1979 Copenhagen Winter School on Abstract Software Specification, Also: Lecture Notes in Computer Science*, Vol. 80, Dines Bjorner, Ed., pp. 292-332, Springer Verlag, NY, 1980.
- [Cheatham89] Thomas E. Cheatham, Jr., "Reusability Through Program Transformations", in *Software Reusability Vol. I*, Ted J. Biggerstaff and Alan J. Perlis, Eds., pp. 321-327, Addison-Wesley Publishing Company, NY, 1989.
- [Freeman87] Peter Freeman and Ruben Prieto-Diaz, "Classifying Software for Reusability", *IEEE Software*, pp. 6-16, January 1987.
- [Gaudel87] M.C. Gaudel and T. Moineau, "A Theory of Software Reusability", *Lecture Notes in Computer Science*, Vol. 300, H. Ganziner, Ed., Springer Verlag, NY, 1987.
- [Goguen87] Joseph A. Goguen, "Reusing and Interconnecting Software Components", in *The IEEE Tutorial on Software Reusability*, Peter Freeman, Ed., pp. 251-263, IEEE Computer Society Press, 1987.
- [Goguen89] Joseph A. Goguen, "Principles of Parameterized Programming", in *Software Reusability Vol. I*, Ted J. Biggerstaff and Alan J. Perlis, Eds., pp. 159-226, Addison-Wesley Publishing Company, NY, 1989.
- [Halstead77] Maurice H. Halstead, *Elements of Software Science*, Elsevier North-Holland Inc., NY, 1977.
- [Heller91] Dan Heller, *Motif Programming Manual*, O'Reilly & Associates, Inc., CA, 1991.
- [Krueger92] Charles W. Krueger, "Software Reuse", *ACM Computing Surveys*, Vol. 24,

No. 2, pp. 131-179, June 1992.

- [Leblang85] David B. Leblang and Gordon D. Mclean, Jr. , "Configuration Management for Large-Scale Software Development Efforts", *Proceedings of the Workshop on Software Engineering Environments for Programming-in-the-Large*, pp. 122-127, Harwichport, MA, 1985.
- [Mclennan93] Michael J. Mclennan, BLT, Copyright (c) 1993 AT&T Bell Laboratories.
- [Matsumoto89] Allen S. Matsumoto and Steven D. Litvintchouk, "Design of ADA Systems Yielding Reusable Components: An Approach Using Structured Algebraic Specification", in *Software Reusability Vol. I*, Ted J. Biggerstaff and Alan J. Perlis, Eds., pp. 227-244, Addison-Wesley Publishing Company, NY, 1989.
- [McIlroy69] M. D. McIlroy, "Mass-Produced Software Components", in *Software Engineering Concepts and Techniques*, pp. 88-98, Brussels 39, Belgium: Petrocelli/Charter, 1969. Paper presented at the 1969 NATO Conference on Software Engineering.
- [Miller89] David B. Miller, Robert Stockton, and Charles W. Krueger, "An Inverted Approach to Configuration Management", *Proceedings of the 2nd International Workshop on Software Configuration Management*, Princeton, NJ, also in *ACM Software Engineering Notes*, pp. 1-4, Vol. 17, No. 7, November 1989.
- [Neighbors89] James M. Neighbors, "DRACO: A Method for Engineering Reusable Software Systems", in *Software Reusability, Vol. I*, Ted J. Biggerstaff and Alan J. Perlis, Eds., pp. 295-320, NY, 1989.
- [Preito-Diaz89] Ruben Prieto-Diaz, "Classification of Reusable Modules", in *Software Reusability, Vol. I*, Ted J. Biggerstaff and Alan J. Perlis, Eds., pp. 99-124, Addison-Wesley Publishing Company, NY 1989.
- [Rochkind75] Marc J. Rochkind, "The Source Code Control System", *IEEE Transactions on Software Engineering*, pp. 364-370, Vol. SE-I, No. 4, December 1975.
- [Sequent90] DYNIX/ptx User's guide, Sequent Computers, Inc., 1990.
- [Swanson92] J.E. Swanson and M.H. Samadzadeh, "A Reusable Software Catalog Interface", *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, pp. 1076-1082, Kansas City, MO, March 1992.
- [Tichy85] Walter F. Tichy, "RCS - A System for Version Control", *Software-Practice and Experience*, pp. 637-654, Vol. 15, No. 7, July 1985.

[Tracz87] Will Tracz, "Software Reuse: Motivators and Inhibitors", *IEEE Tutorial on Software Reuse: Emerging Technology*, Peter Freeman, Ed., pp. 251-263, IEEE Computer Society Press, 1987.

APPENDICES

APPENDIX A

GLOSSARY AND TRADEMARK INFORMATION

GLOSSARY:

Baseline: A shared project database item. It has all the components from which the final product can be created. Once a component is placed in the baseline, it is not usually modified.

Bound Configuration Thread: A set of versions of modules constituting a software system.

Configuration: The configuration of a software system is the list of modules the software is made up of plus their interconnection architecture.

Configuration Management: The art of identifying, organizing, and controlling modifications to a software being built.

Configuration Thread: The versions of modules constituting a software system.

Delta: When one version is stored fully, the other versions are represented based on differences from this version; the differences are called deltas.

Option Letter: An alphabetic character attached to a delta in RCS that specifies the inclusion of that delta as a command line argument while making the file corresponding to a higher version.

Piece-Table: A one-dimensional array on which all operations are performed in RCS.

RCS: Revision control system.

Release: An assortment of the existing versions of software modules that meet the requirements of a software being built, partially or fully.

Revision: A revision is a new version intended to supersede an older version.

SCCS: Source code control system.

State: A variable associated with a version of a software module in RCS that indicates the status of the version, i.e., released, experimental, or stable.

Version: An alternative form of a software module made to provide improvements, adapt to different environments, or mark a new release.

TRADEMARK INFORMATION:

DYNIX/ptx: A registered trademark of Sequent Computer System, Inc.

Motif: A registered trademark of the Open Software Foundation.

OSF: A registered trademark of the Open Software Foundation.

Sequent Symmetry S/81: A registered trademark of the Sequent Computer System, Inc.

Sun: A registered trademark of Sun Microsystems.

X Window System: A registered trademark of the Massachusetts Institute of Technology.

UNIX: A registered trademark of AT&T.

APPENDIX B

USER GUIDE

Version Control Interface (VCI) is an interface to a software library system. It is built on top of RCS (Revision Control System). It provides for efficient storage of files. In VCI, the concepts of version management and software classification are combined in an effort to provide a software cataloging and storage system. The various functionalities of VCI are described below.

1. CHECK-IN:

The check-in process is done in two phases.

Phase I:

1. Push the "Libin" Button.
2. Give the name of the file to be checked-in.
3. Give the description of the file.
4. Select the method of determining the file position in the library.
5. If the method of selecting the file position is "Position selected by system", then
Phase I of checking-in a file is complete.
6. If the method of selecting the file position is "Position suggested by user", then
select the file name to which the new file (that is to be checked-in) is to be
attached.

This completes Phase I.

Description of Step 3:

The method used to describe files is similar to that given by Prieto-Diaz [Prieto-Diaz89]. Some of the attributes to be described are mentioned below.

Function: The specific action performed by a program (or a file in general).

Method: A description of the objects manipulated by the program, which is the data structures used by the program. For example, <lines, B-Tree>, <character, array>, or <integers, array>.

Implementation Details: The following descriptions must be provided for each file

1. Functional Area, describing the area of the application of the file.
2. Location or Setting, describing where the file was intended to be used.

Example: <Database Management, Catalog Sales>

Description of Step 6:

To help a user select the appropriate file, the following aids are provided.

Logical Link: The user can type any meaningful word that summarizes the function of the file being checked in. All the files linked logically to this word pop up in a list dialog, enabling the user to connect the file being checked in to a file performing a similar function.

Thesaurus Option: If the above fails to produce any results, the user can use the thesaurus option. When the user pushes the thesaurus button, the user is provided with equivalent words in use in the library to describe the same function, if any. For example, delete and remove are synonyms. There may be many files connected to "delete" but if "remove" is

tried, the user may come up with no files. Using the thesaurus option, the user can try "delete" next in order to get the files connected to "delete".

Description: Double-clicking on a file name in the list gives the description of that file.

Tree of the library structure: The tree of the library structure can be seen by pressing the "Display" button of the main menu. Any file name or head can be given to see the tree from the particular file. Navigation is possible in the zoom-in mode. Only three children connected to the current file are shown at a time. The other file(s) connected can be seen by clicking on the top or bottom arrowhead in the current file box, as needed. A step-by-step zoom-out facility is provided to judge the depth and breadth of the tree at the desired position.

Phase II:

Once the user has checked-in a file, logical links to all the words, which can describe the function performed by the file checked-in, must be provided. This is done by pushing the "Link" button in the main menu.

2. CHECK-OUT:

1. Push the "Libout" button in the main menu.
2. Select the method of checking-out.
3. If the method of checking-out is by "filename", all methods described in the selection of a link file name (Step 6 of check-in) can be used.
4. If "Pattern search" is selected, a word can be given which is searched for in the description of the files, and all files with that given word are presented in the form of a list.

5. The selection of the file from the List completes the process of checking-out a file.

3. DELETION:

Files can be deleted from the library using the "Deletion" option.

4. STATS:

The performance of the library can be seen by clicking on the "Stats" button. The disk space utilization graph and the graph for frequency of check-outs can be viewed on the screen. To view the graphs using BLT, the blt-wish and the TCL library must be installed, and their paths must be present in the user's **.profile**. A sample of the changes to be incorporated in the **.profile** of the users for the Sequent S/81 System of Computer Science Department at OSU is as follows.

```
path = .....:/contrib/bin:.....:/home directory path
TCL_LIBRARY = /contrib/lib/tcl
TK_LIBRARY = /contrib/lib/tk
export .....TCL_LIBRARY TK_LIBRARY
```

NOTE:

1. Only on approval by the library administrator is a file checked-in by any user actually gets checked into the library. Until then, the added file is maintained as an experimental file. All operations explained above can be performed on all experimental files as well.
2. The delete and move operations are affected only after the library administrator

approves them. Hence a delete or move operation is not immediately visible to the users.

APPENDIX C

LIBRARY ADMINISTRATOR GUIDE

1. Description

VCI is an interface to building and maintaining a software repository. VCI requires RCS, X Windowing system, Motif, and BLT to be installed.

2. Installing VCI

Create a directory, henceforth called the library directory, to store the library files. Initialize the variable `lib_d` in `main.c` to the path of this directory. Compile the program using the makefile provided. Create another directory, henceforth called the executable directory, to store the executable file. Both these directories should be accessible to the users. Store all the bitmap files in the library directory. Initialize the various files as described in Section 3 of this document. BLT must be accessible to all users. The corresponding changes in the user's `.profile`, if needed, must be advised by the administrator. A sample of the required changes for the Sequent S/81 System of Computer Science Department at OSU is shown in the USER GUIDE. If BLT or the Tcl library, which is needed to run BLT, is not accessible, the quit and save buttons will not be active in the graphs. Adapt the USER GUIDE for the local configuration.

3. System Files Associated with VCI

The various system files (files used to maintain the system) are described below.

Password.dat: This file is used to hold the different passwords accepted by the system. Only one password must be given per line. The case of letters is significant. Special characters can be used to form a password. The length of the password is restricted to fifty characters. The password of the library administrator is not given in this file.

Default.dat: The first line in this file is the password of the system administrator. The second line is the path of the library directory. If the library directory has to be changed, this can be done by changing the corresponding line in the default.dat file in the original library directory. This feature is useful in changing the library directory without having to recompile the program.

Main.dat: This file records the essential information of each file in addition to the library structure. The library structure is implicitly recorded in the order of the files in this file. First the name of the file is recorded. Next, the RCS filename in which the file is stored is recorded. The logical name in the RCS file is recorded next. The description of the file is followed by the space savings factor. In this way, the data of each file in a preorder traversal of the tree is recorded and a NULL is placed when a file is not encountered in the preorder traversal. In this way, the tree of the library structure is implicitly stored. When VCI is executed, the tree is reconstructed from the order of the files in the main.dat file. The structure of this file is important for the correct functioning of VCI. A sample of this file is provided in Figure 20.

```

Makefile          /* Name of the file */
Makefile,v       /* RCS file name */
rcsMakefile      /* Logical name in the RCS file */
nadella          /* Name of the user who checked-in the file */
.
nadella          /* E-mail address of the user */
.
blt ver1.5       /* Function */
.
Method not known /* Method */
.
Implementation details not known /* Implementation details */
.
0                /* Number of times checked out */
.
-11.003111       /* space saved (%)* */
.
-256             /* Total space saved */
.
1                /* Status of file */
.
NULL
NULL

```

Figure 20. Structure of the main.dat file

Experimental.dat: This file records the request of the user to include a file in the library. In addition to recording the name of the user file, this file also records the description of the user file and the file to which it has to be connected. The user file to be checked-in will be searched for in the library directory. The structure of the experimental.dat file is important for the proper working of the system. All interactions with this file should be done only through the system. A sample of this file is shown in Figure 21.

```

rcsmarif_try          /* Name of the file*/
Arif Muljadi          /* Name of the user who checked-in the file */
.
marif@a.cs.okstate.edu /* E-mail address of the user */
.
libin testing         /* Function */
.
any method            /* Method */
.
detail1               /* Implementation details */
detail2
.
0.000000             /* Total space saved */
.
0                     /* Space saved as a percentage */
.
0                     /* Status of the file */
.

```

Figure 21. Structure of the experimental.dat file

Update.dat: This file records the request of the user to delete a file from the library, move a file, or change the description of a file. The necessary information to perform the above operations, in addition to the VCI password of the user requesting the same, is recorded. A sample of this file is presented in Figure 22.

```

DELETE FILE dyarray.c ,requested by user
Move File marif1 to a child of head requested by user
NEW SYMBOLIC LINK FROM marif_try_link to head with distance 2 added by
user
Move File marif1 to a child of head requested by user
Move File dyarray.c to a child of head requested by user
DELETE FILE dyarray.c ,requested by user
NEW SYMBOLIC LINK FROM copy to dyarray.c with distance 25 added by user
NEW SYMBOLIC LINK FROM copy to dyarray.c with distance 25 added by user
NEW SYMBOLIC LINK FROM move to dyarray.c with distance 10 added by user

```

Figure 22. Structure of the update.dat file

Link.dat: This file records the various links of the system. A sample structure of the file is shown in Figure 23. This file records the name of the link, the name of the file to which it is connected, the status (if experimental), and the distance. This file can be edited to change or update links.

```
marif_try_link head 1 2 copy dyarray.c 1 25 move dyarray.c 1 10
```

Figure 23. Structure of link file

Thesaurus.dat: This file is the data file for the thesaurus of the system. The structure of this file is shown in Figure 24. This file can also be edited to make changes.

```
move                /* Word */
copy delete increment . /* Equivalent words */
create
make originate move .
copy
create delete move .
kill
destroy delete .
.
```

Figure 24. Structure of the thesaurus.dat file

4. Maintaining the System

The library administrator, in addition to setting up the system, has the following responsibilities.

Check-in Experimental Files:

The Library administrator has to check-in or discard the experimental files depending on their merit. The experimental files can be checked in by going through the check-in process after logging into the system using the library administrator's password. The file must first be discarded from the Experimental.dat file before trying to add it to the system. The original file from the library directory should be erased after the file is checked-in.

Update the System:

The requests of users to delete a file, move a file, or change the description of a file, are recorded in the update.dat file. The necessary changes once done should be made by logging in as the library administrator. The update.dat file must be initialized once the requests have been processed.

5. Observations

The user cannot change any system files through VCI. Only the library administrator can do so. It is the responsibility of the library administrator to maintain the integrity of the system files. A backup of system files can be kept in a different directory to recover from unforeseen errors. If a situation arises, where the main.dat file is lost, then the only alternative is to check out all the files from the RCS files one by one, and rename them. The library can be created from scratch by checking these files in again. The loss of update.dat and experimental.dat is not so critical because the users can in principle repeat their request on being informed of the loss.

Appendix D

PROGRAM LISTING

The program files are presented in this appendix. The order of the following listing of modules, following a typical order of usage of the functionalities and features in the user interface of VCI, is as follows.

- Main.c
- File_select.c
- Description.c
- Position.c
- User_select.c
- Libout.c
- Pattern.c
- Link.c
- Display.c
- Show.c
- Delete.c
- Move.c
- Options.c
- Stat.c
- Help.c
- Exit.c
- Lock.c
- Load.c
- List.h
- List.c
- Thesaurus.h
- Thesaurus.c


```

/*
* * * * *
*   Filename : main.c
*
*   Programmed by : N. Sunil Chakravarthy
*
*   Last updated on : 08.17.94
* * * * *

This file contains the code for initial screen and the screen 1 containing the drawn
buttons.
On pressing the drawn buttons, the respective functions are called.

*/

#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/PushB.h>
#include <Xm/FileSB.h>
#include <Xm/DialogS.h>
#include <sys/stat.h>
#include <X11/Xos.h>
#include <Xm/DrawingA.h>
#include <Xm/DrawnB.h>
#include <Xm/Text.h>
#include <Xm/LabelG.h>
#include <Xm/RowColumn.h>
#include <Xm/Form.h>
#include <Xm/ToggleBG.h>
#include <Xm/SeparatoG.h>
#include <Xm/PushBG.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <ctype.h>
#include <errno.h>
#include <Xm/VendorE.h>
# include <X11/cursorfont.h>

#define col 1

/* The initial directory of the library is stored here. */
#define library "/m/nadella/image/library/"

extern load();
extern load_file();
extern file_sel();
extern lib_out();
extern display();
extern delete();
extern move();
extern link();
extern write_file();
extern writelist();
extern stat_button();
extern errordialog();
extern exitdialog();
extern file_lock();
extern util();
extern readme();
extern load_help();
extern map();
extern destroy();

extern int load_time;
extern int mode;

extern struct list *listhead;

/* This structure is used to hold the pixmap and other information of the drawn
buttons.
*/

```

```

typedef struct {
    Widget drawn_w;
    Widget popup;
    char *pixmap_file;
    char *pixmap;
    Pixmap pix;
    Pixmap pixb;
    int pid;
} ExecItem;
typedef struct {
    char *exec_argv[10];
} cin_item;
cin_item cin_exec[] = {
    { NULL }
};

void drawing_callback();
void tree();
void row_help();
void warn_dialog();

int xpos, ypos;
extern char lib_d[100], src_d[100];

struct file_name *curr;
struct children *pt, *pt1, *pt2, *pt3;

/* Store the name of the pixmaps of the drawn buttons. */
ExecItem prog_list[] = {
    { NULL, NULL, "libinP", "libin", NULL, NULL, , 0 },
    { NULL, NULL, "liboutP", "libout", NULL, NULL, , 0 },
    { NULL, NULL, "displayP", "display", NULL, NULL, , 0 },
    { NULL, NULL, "linkP", "link", NULL, NULL, , 0 },
    { NULL, NULL, "moveP", "move", NULL, NULL, , 0 },
    { NULL, NULL, "deleteP", "delete", NULL, NULL, , 0 },
    { NULL, NULL, "optionP", "option", NULL, NULL, , 0 },
    { NULL, NULL, "statsP", "stats", NULL, NULL, , 0 },
    { NULL, NULL, "helpP", "help", NULL, NULL, , 0 },
    { NULL, NULL, "quitP", "quitP", NULL, NULL, , 0 },
};

void do_search(), new_file_cb(), file_exit(), initial();
XmStringCharSet charset = XmSTRING_DEFAULT_CHARSET;

XtAppContext app; /* application context for the whole program */
GC gc; /* used to render pixmaps in the widgets */
XtAppContext app; /* application context for the whole program */
GC gc; /* used to render pixmaps in the widgets */
static void reset(), reset_btn(), redraw_button(), exec_prog(), cin_input(),
cin_pushed(), cin_dialog_pushed();
void cin_R_dialog(), cin_Rcallback(), cin_R_dialog_ok();
void cout_R_dialog(), cout_Rcallback(), cout_R_dialog_ok(), cout_dialog_pushed(),
cout_pushed();

void load_defaults();
void get_fcsfile(), rcsfile_ok();

void attrib_pushed(), toggled(), check_bits();
void stat_ok();
void check_passwd(), next_menu();
char pwd[80];
char *passwd;
char rcs_file[20];
char preference[200];

Widget toplevel, rowcol, temp;
char str[200];
int sys_adm = 0;
int pushed = -1;

XtEventHandler drawfontscreen();
unsigned long toggles_set;
Widget form1, draw_w, cin_pull;

```

```

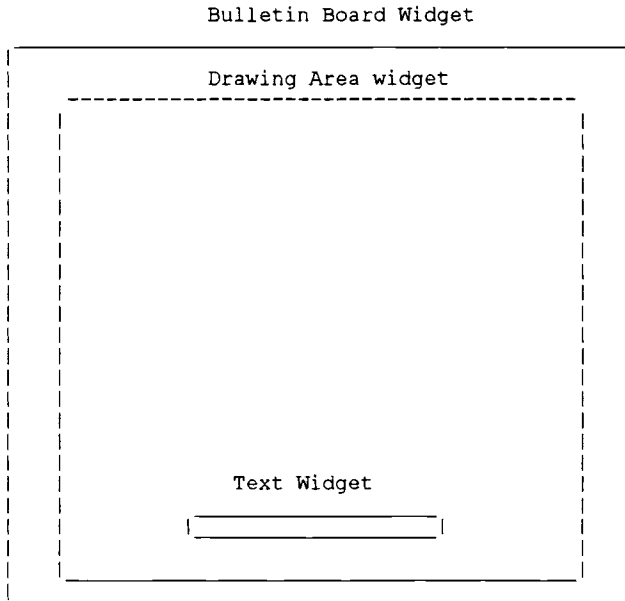
/*
  For the first screen, create a toplevel shell of size 700 by 700. Create a drawing area
  widget as a child of this toplevel shell. Create a text widget as a child of the drawing
  area widget to accept the password. Link a exposure callback to the drawing area widget to
  write the text on being exposed. create the pixmap for the icon. Finally, load the
  defaults, tree and help information.
*/

```

```

/* The structure of this dialog is as follows.

```



```

*/
main(argc, argv)
int    argc;
char   *argv[];
{
    Widget text_w;
    Pixmap pixmap;
    Pixel fg, bg;
    Window window, root;
    unsigned int width, height, border, depth;
    int    x, y;

    toplevel = XtVaAppInitialize(&app, "Demos", NULL, 0, &argc, argv,
        NULL, XmnWidth, 700, XmnHeight, 700, XnmwmInputMode,
        1, XmnminHeight, 100, XmnminWidth, 100, XmnTitle, "Version Control
        Interface",
        XmnIconName, "Version Control Interface", XmnInitialState,
        IconicState, NULL);

    /* Loading the defaults, tree and help information. */

    strcpy(lib_d, library);
    load_defaults();
    load();
    load_file();
    load_help();

    draw_w = XtVaCreateManagedWidget("dbutton", xmDrawingAreaWidgetClass,
        toplevel, XmnWidth, 700, XmnHeight, 700,
        XmnborderWidth, 0, XmnshadowThickness, 0, NULL);

    XtVaGetValues(draw_w, Xmnforeground, &fg, Xmnbackground, &bg, NULL);
    gc = XCreateGC(XtDisplay(draw_w), RootWindowOfScreen(XtScreen(toplevel)),
        NULL, 0);
    XSetForeground(XtDisplay(toplevel), gc, fg);

```

```

XSetBackground(XtDisplay(toplevel), gc, bg);

/* Setting the icon of the application. */

sprintf(str, "%svciicon", lib d);
pixmap = XmGetPixmap(XtScreen(toplevel), str, fg, bg);
XGetGeometry(XtDisplay(toplevel), pixmap, &root, &x, &y, &width,
             &height, &border, &depth);

window = XCreateWindow(XtDisplay(toplevel), root, 0, 0, 100, 100,
                      5, (unsigned)0, CopyFromParent, CopyFromParent, CopyFromParent,
                      CopyFromParent);
XtVaSetValues(toplevel, XmNiconWindow, window, NULL);
XSetWindowBackgroundPixmap(XtDisplay(toplevel), window, pixmap);
XClearWindow(XtDisplay(toplevel), window);

text_w = XtVaCreateManagedWidget("text w", xmTextWidgetClass, draw_w,
                                 XmNx, 350, XmNy, 650, NULL);

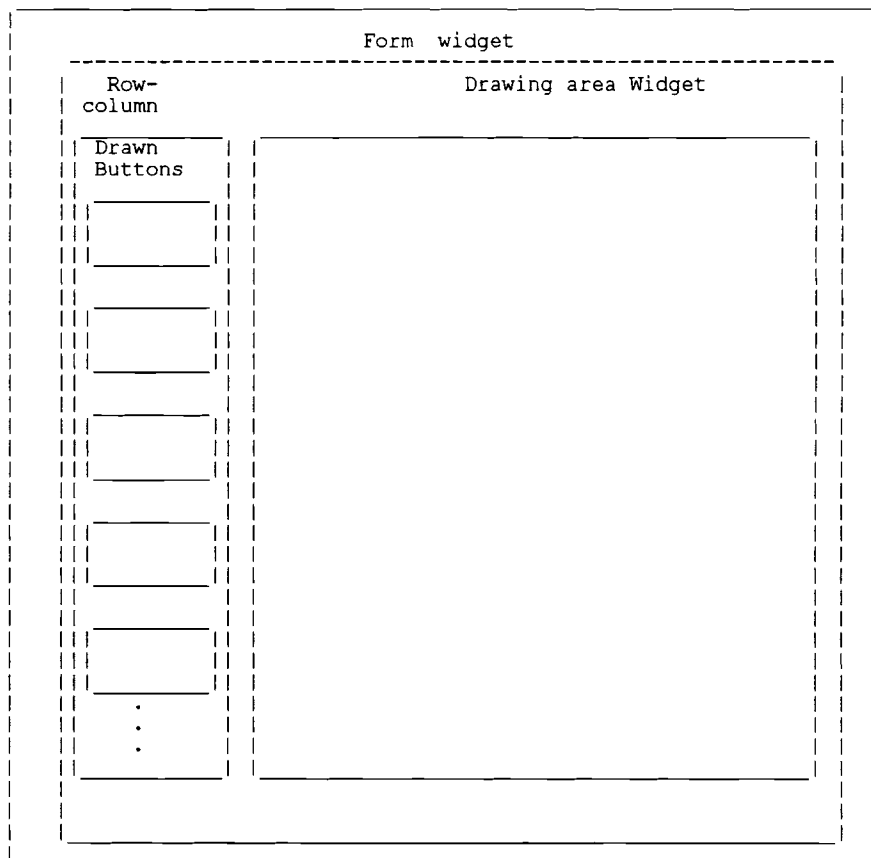
XtAddCallback(text_w, XmNmodifyVerifyCallback, check_passwd, NULL);
XtAddCallback(text_w, XmNactivateCallback, check_passwd, NULL);

/* Setting the exposure callback. */

XtAddEventHandler(draw_w, ExposureMask, False, drawfontscreen, NULL);

XtRealizeWidget(toplevel);
XtAppMainLoop(app);
}
/* This function is used to create the second screen of the application.
   Create a form widget as a child of toplevel shell. Connect a rowcolumn
   widget to the left and a drawing area widget to the right. Create the
   drawn buttons as children of the rowcolumn widget. Create the respective
   pixmaps from the structure and and store it in the structure.
*/
/* The structure of this dialog is as follows.
   Bulletin Board Widget

```



```

*/

void next_menu()
{
    Pixmap pixmap;
    Pixel fg, bg;
    Arg args[1];
    XmString one, two, three, four, five;
    int i;
    Cursor cursor;

    /* Create the form Widget. */

    form1 = XtVaCreateManagedWidget("form", xmFormWidgetClass, toplevel,
        NULL);

    /* Create the rowcolumn widget. */

    rowcol = XtVaCreateManagedWidget("rowcol", xmRowColumnWidgetClass,
        form1, XmNnumColumns, col, XmNpacking, XmPACK_COLUMN, XmNorientation,
        XmVERTICAL, XmNtopAttachment, XmATTACH_FORM, XmNleftAttachment,
        XmATTACH_FORM, NULL);
    mode = 0;

    /* Add the help callback. This will display the initial help (readme) if the F1
       key is pressed in this screen. */

    XtAddCallback(rowcol, XmNhelpCallback, row_help, 1);

    /* Create the drawing area Widget. */

    draw_w = XtVaCreateManagedWidget("dbutton", xmDrawingAreaWidgetClass,
        form1, XmNtopAttachment, XmATTACH_FORM, XmNbottomAttachment,
        XmATTACH_FORM, XmNleftAttachment, XmATTACH_WIDGET,
        XmNleftWidget, rowcol,
        XmNwidth, 700, XmNheight, 700,
        XmNborderWidth, 2, XmNshadowThickness, 1,
        NULL);

    /* Get the foreground and background colors of the rowcol widget
       so that the gc (DrawnButtons) will use them to render pixmaps.
    */

    XtVaGetValues(rowcol, XmNforeground, &fg, XmNbackground, &bg, NULL);
    gc = XCreateGC(XtDisplay(rowcol), RootWindowOfScreen(XtScreen(rowcol)),
        NULL, 0);
    XSetForeground(XtDisplay(rowcol), gc, fg);
    XSetBackground(XtDisplay(rowcol), gc, bg);
    cursor = XCreateFontCursor(XtDisplay(toplevel), 58);
    XDefineCursor(XtDisplay(toplevel), XtWindow(toplevel), cursor);

    /* Setting the icon of the application. */

    pixmap = XmGetPixmap(XtScreen(rowcol), "vertical", fg, bg);
    XtVaSetValues(rowcol, XmNbackgroundPixmap, pixmap, NULL);

    /* Create the required number of drawn buttons as children of the rowcolumn widget
    */

    for (i = 0; i < XtNumber(prog_list); i++) {

        /* The pixmap is taken from the name given in the structure. */

        sprintf(str, "%s%s", lib_d, prog_list[i].pixmap_file);
        prog_list[i].pix = XmGetPixmap(XtScreen(rowcol), str, fg, bg);

        sprintf(str, "%s%s", lib_d, prog_list[i].pixmap_file);
        prog_list[i].pixb = XmGetPixmap(XtScreen(rowcol), str, bg,
            fg);

        /* Create a drawn button 64x64 (arbitrary, but sufficient).
           * ShadowType has no effect till pushButtonEnabled is false.
        */
    }
}

```

```

*/
prog_list[i].drawn_w = XtVaCreateManagedWidget("dbutton",
        xmDrawnButtonWidgetClass, rowcol, XmNwidth, 75,
        XmNheight, 65, XmNhighlightOnEnter, True,
        XmNhighlightThickness, 5, XmNpushButtonEnabled, True,
        XmNshadowType, XmSHADOW_ETCHED_OUT, NULL);

temp = prog_list[0].drawn_w;

/* If this button is selected, execute the exec_prog function. */
XtAddCallback(prog_list[i].drawn_w, XmNactivateCallback,
        exec_prog, &prog_list[i]);

/* When the resize and expose events come, redraw the pixmap. */
XtAddCallback(prog_list[i].drawn_w, XmNexposeCallback, redraw_button, i);
/* XtAddCallback(draw_w,XmNinputCallback,drawing_callback,NULL);*/
XtAddCallback(prog_list[i].drawn_w, XmNresizeCallback, redraw_button, i);
}
}

/* This function is used to draw the text in the initial screen when it is exposed */
XtEventHandler drawfontscreen(widget_id, client_data, call_data)
Widget widget_id;
caddr_t client_data;
caddr_t call_data;
{
    Window window = XtWindow(widget_id);
    GC myGC;
    Pixel fg, bg;
    Pixmap pixmap;
    Display * display;

    /* Get the gc, display, foreground, and background. */

    display = XtDisplay(widget_id);
    myGC = XCreateGC(XtDisplay(widget_id), RootWindowOfScreen(XtScreen(widget_id)
        ), NULL, 0);
    XtVaGetValues(widget_id, XmNforeground, &fg, XmNbackground, &bg,
        NULL);
    XSetForeground(XtDisplay(widget_id), myGC, fg);
    XSetBackground(XtDisplay(widget_id), myGC, bg);

    /* Get the pixmap to shade the screen. */

    sprintf(str, "%sfront", lib_d);
    pixmap = XmGetPixmap(XtScreen(widget_id), str, bg, fg);
    XSetStipple(display, myGC, pixmap);
    XSetFillStyle(display, myGC, FillStippled);
    XFillRectangle(display, window, myGC, 0, 0, 700, 700);
    XSetFillStyle(display, myGC, FillSolid);

    XDrawString(display, window, myGC, 200, 185, "VERSION CONTROL INTERFACE", 27);
    XDrawString(display, window, myGC, 340, 300, "BY", 3);

    XDrawString(display, window, myGC, 270, 350, "SUNIL C NADELLA", 15);
    XDrawString(display, window, myGC, 330, 375, "AND", 4);
    XDrawString(display, window, myGC, 270, 400, "Dr.M.H.SAMADZADEH", 17);
    XDrawString(display, window, myGC, 220, 500, "COMPUTER SCIENCE DEPARTMENT", 27);
    XDrawString(display, window, myGC, 230, 550, "OKLAHOMA STATE UNIVERSITY", 26);
    XDrawString(display, window, myGC, 220, 675, "Password:", 9);
}

/* redraw_button() draws the pixmap (client_data) into its DrawnButton
 * using the global GC (gc). Get the width and the height of the pixmap
 * being used so that we can either center it in the button or clip it.
 */

static void
redraw_button(button, num, cbs)
Widget button;

```

```

int num;
XmDrawnButtonCallbackStruct *cbs;
{
    int srcx, srcy, destx, desty, pix_w, pix_h;
    int drawsize, border;
    Dimension bdr_w, w_width, w_height;
    short hlthick, shthick;
    Window root;
    Pixmap pixmap;

    if (num == pushed)
        pixmap = prog_list[num].pibx;
    else
        pixmap = prog_list[num].pix;

    /* Get the width and the height of the pixmap. */
    XGetGeometry(XtDisplay(button), pixmap, &root, &srcx, &srcx, &pix_w,
                 &pix_h, &srcx, &srcx);

    /* Get the values of all the resources that affect the entire
       geometry of the button.
    */
    XtVaGetValues(button, XmNwidth, &w_width, XmNheight, &w_height, XmNborderWidth,
                  &bdr_w, XmNhighlightThickness, &hlthick, XmNshadowThickness, &shthick,
                  NULL);

    /* Calculate the available drawing area, width 1st. */
    border = bdr_w + hlthick + shthick;

    /* If window is bigger than pixmap, center it; else clip pixmap. */
    drawsize = w_width - 2 * border;
    if (drawsize > pix_w) {
        srcx = 0;
        destx = (drawsize - pix_w) / 2 + border;
    } else {
        srcx = (pix_w - drawsize) / 2;
        pix_w = drawsize;
        destx = border;
    }

    /* Now the height ... */
    drawsize = w_height - 2 * border;
    if (drawsize > pix_h) {
        srcy = 0;
        desty = (drawsize - pix_h) / 2 + border;
    } else {
        srcy = (pix_h - drawsize) / 2;
        pix_h = drawsize;
        desty = border;
    }

    XCopyArea(XtDisplay(button), pixmap, XtWindow(button), gc, srcx,
              srcy, pix_w, pix_h, destx, desty);
}

/* The button has been pressed; identify which button has been
 * pressed, change its pixmap (to show it is pressed) and call the respective
 * function.
 */

static void
exec_prog(drawn_w, program, cbs)
Widget drawn_w;
ExecItem *program;
XmDrawnButtonCallbackStruct *cbs;
{
    Widget dialog, cin_pull, Toggle, cin_R, cin_N, cin_S, cin_T, cin_M, rowcoll;
    XmString one, two, three, four, five;
    Arg args[5];
    int i, temp;
    char str[80];
    extern void exit();
}

```

```

temp = pushed;
pushed = -1;

/* Reset the previous pressed button to normal pixmap. */
if (temp != -1)
    redraw_button(prog_list[temp].drawn_w, temp, NULL);

/* Check which button has been pressed and call the respective function
 * after changing the pixmap.
 */

if (drawn_w == prog_list[0].drawn_w) {
    /* Libin has been pushed. */

    pushed = 0;
    redraw_button(prog_list[0].drawn_w, 0, NULL);
    memset(src_d, '\0', 100);
    strcpy(src_d, "./");
    file_sel(rowcol);
    return;
}
if (drawn_w == prog_list[1].drawn_w) {
    /* Libout has been pushed. */

    pushed = 1;
    redraw_button(prog_list[1].drawn_w, 1, NULL);
    memset(src_d, '\0', 100);
    strcpy(src_d, "./");
    lib_out(prog_list[1].drawn_w);
    return;
}
if (drawn_w == prog_list[2].drawn_w) {
    /* Display has been pushed. */

    pushed = 2;
    redraw_button(prog_list[2].drawn_w, 2, NULL);
    memset(src_d, '\0', 100);
    strcpy(src_d, "./");
    display(prog_list[0].drawn_w);
    return;
}
if (drawn_w == prog_list[3].drawn_w) {
    /* Link has been pushed. */

    pushed = 3;
    redraw_button(prog_list[3].drawn_w, 3, NULL);
    memset(src_d, '\0', 100);
    strcpy(src_d, "./");
    link(prog_list[3].drawn_w);
    return;
}
if (drawn_w == prog_list[4].drawn_w) {
    /* Delete has been pushed. */

    pushed = 4;
    redraw_button(prog_list[4].drawn_w, 4, NULL);
    memset(src_d, '\0', 100);
    strcpy(src_d, "./");
    move(prog_list[4].drawn_w);
    return;
}
if (drawn_w == prog_list[5].drawn_w) {
    /* Move has been pushed. */

    pushed = 5;
    redraw_button(prog_list[5].drawn_w, 5, NULL);
    memset(src_d, '\0', 100);
    strcpy(src_d, "./");
    delete(prog_list[0].drawn_w);
    return;
}
}

```



```

if (drawn_w == prog_list[6].drawn_w) {
    /* Options has been pushed. */

    pushed = 6;
    redraw_button(prog_list[6].drawn_w, 6, NULL);
    memset(src_d, '\0', 100);
    strcpy(src_d, "./");

    /* Give choices for option and proceed. */

    util(rowcol);
    return;
}
if (drawn_w == prog_list[7].drawn_w) {
    /* Stat has been pushed. */

    pushed = 7;
    redraw_button(prog_list[7].drawn_w, 7, NULL);
    memset(src_d, '\0', 100);
    strcpy(src_d, "./");

    /* Give a warning to change the .profile and proceed. */

    warn_dialog();
    return;
}
if (drawn_w == prog_list[8].drawn_w) {
    /* Help has been pushed. */

    pushed = 8;
    redraw_button(prog_list[8].drawn_w, 8, NULL);

    /* Give choices for help. */

    get_help();
    return;
}
if (drawn_w == prog_list[9].drawn_w) {
    /* Exit has been pushed. */

    pushed = 9;
    redraw_button(prog_list[9].drawn_w, 9, NULL);
    memset(src_d, '\0', 100);
    strcpy(src_d, "./");

    /* Write main.dat file if user is system_administrator. */

    if (sys_adm)
        write_file();

    /* Confirm intention to exit and proceed accordingly. */

    exitdialog();
}
}

/*
 * This function is called when a password is typed in the text widget in screen1.
 * Note that this function is called after each character is typed. The characters
 * are stored in the variable pwd and is checked against those stored in the
 * password file once the return key is pressed. If a wrong password is entered
 * the text widget is cleared and a warning dialog is popped up.
 */

void
check_passwd(text_w, unused, cbs)
Widget      text_w;
XtPointer   unused;
XmTextVerifyCallbackStruct *cbs;
{
    char      *new, *word;
    int       len;
    FILE      *fp;

```

```

char   buf[82];
int    verified = 0;

if (cbs->reason == XmCR_ACTIVATE) {

    /* Enter key is preessed check password. */

    sprintf(str, "%swd.out", lib_d);
    if ((fp = fopen(str, "rt")) == NULL) {
        printf("unable to open password file \n");
        exit(0);
    }
    memset(buf, '\0', 80);
    fgets(buf, 80, fp);
    while (!feof(fp)) {
        word = strtok(buf, "\n");

        /* Check password against those in the password file. */

        if (strcmp(passwd, word) == 0) {

            /* Yes, the password is in password file. Now check
             * If it is the system administrator. If so, set sys_adm to 1.
             * This is used throughout the application.
             */

            if (strcmp(passwd, pwd) == 0)
                sys_adm = 1;
            XtDestroyWidget(draw_w);
            verified = 1;

            /* Display the second screen as the password has verified.
             */

            next_menu();
            return;

        }
        fgets(buf, 80, fp);
    }
    fclose(fp);

    /* Wrong password was entered. Clear the text widget and give warning. */

    XmTextSetString(text_w, NULL);
    if (passwd)
        passwd[0] = NULL;
    errorDialog("not correct password");
    return;
}

if (cbs->text->ptr == NULL) {
    cbs->endPos = strlen(passwd); /* delete from here to end */
    passwd[cbs->startPos] = 0; /* backspace--terminate */
    return;
}

if (cbs->text->length > 1) {
    cbs->doit = False; /* don't allow "paste" operations */
    return; /* make the user *type* the password! */
}

new = XtMalloc(cbs->endPos + 2); /* new char + NULL terminator */
if (passwd) {
    strcpy(new, passwd);
    XtFree(passwd);
} else
    new[0] = NULL;
passwd = new;
strncat(passwd, cbs->text->ptr, cbs->text->length);
passwd[cbs->endPos + cbs->text->length] = 0;

/* "*" is displayed for each character typed so as to not reveal
 * the password while it is being typed in the text widget.
 */

for (len = 0; len < cbs->text->length; len++)
    cbs->text->ptr[len] = '*';
}

```

```

/* This function loads the library administrator password and library directory
 * from the default.dat file.
 */

void load_defaults()
{
    FILE    *fp;
    char    *c;
    sprintf(str, "%sdefault.dat", lib_d);
    fp = fopen(str, "r");
    if (!fp) {
        errordialog("Unable to open defaults file ... ");
        exit(0);
    }
    fgets(str, 80, fp);
    c = strtok(str, "\n");
    strcpy(pwd, c);
    fgets(str, 80, fp);
    c = strtok(str, "\n");
    strcpy(lib_d, c);
}

/* The F1 key is pressed in screen2, show readme file in a dialog. */
void row_help(Widget w_id, XtPointer client_data, XmPushButtonCallbackStruct*cbs)
{
    readme();
}

/* This function is used to popup a warning dialog box to warn the user to
 * change the .profile before trying to view the graphs.
 */

void warn_dialog()
{
    Widget dialog, text_w1, rowcol2, rowcol3, pb1, pb2;
    XmString text, one, two;
    Arg ags[10];
    char    temp[2000];

    /* Create a bulletin board dialog. */

    XtSetArg(ags[0], XmNautoUnmanage, True);
    XtSetArg(ags[1], XmNdefaultPosition, False);
    XtSetArg(ags[2], XmNallowShellResize, True);
    dialog = XmCreateBulletinBoardDialog(toplevel, "WARNING", ags, 3);
    XtAddCallback(dialog, XmNmapCallback, map, 150100);

    /* Create a rowcol widget as a child of bulletin board widget. */
    rowcol2 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, dialog,
        XmNnumColumns, 1, XmNorientation, XmVERTICAL, NULL);

    /* Create a text widget with the required warning. */

    XtSetArg(ags[0], XmNrows, 10);
    XtSetArg(ags[1], XmNcolumns, 60);
    XtSetArg(ags[2], XmNeditable, False);
    XtSetArg(ags[3], XmNeditMode, XmMULTI_LINE_EDIT);
    XtSetArg(ags[4], XmNwordWrap, True);
    XtSetArg(ags[5], XmNvalue, "The .profile needs to be modified as described in
contents of help to view the graphs. \nClick 'Ok' button only if the .profile has
been modified.");
    XtSetArg(ags[6], XmNscrollHorizontal, False);
    text_w1 = XmCreateScrolledText(rowcol2, "text", ags, 7);

    /* Create a form widget to hold "OK" and "CANCEL" buttons. */

    rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcol2, NULL);
    one = XmStringCreateSimple("OK");
    two = XmStringCreateSimple("CANCEL");

    /* Create "OK" and "cancel" buttons. */

    pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,

```

```

        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNheight, 25, NULL);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
        XmNrightOffset, 5, XmNheight, 25, NULL);

/* If the "OK" button is pushed call stat_ok function. If "cancel" is pushed
 * destroy this dialog and proceed as if the stat button was not pushed.
 */

XtAddCallback(pb1, XmNactivateCallback, stat_ok, dialog);
XtAddCallback(pb2, XmNactivateCallback, destRoy, dialog);

XtFree(one);
XtFree(two);
XtManageChild(text_w1);
XtManageChild(rowcol2);
XtManageChild(rowcol3);
XtManageChild(dialog);
}

/* The user has pressed "Ok" in the above warning dialog and wants to proceed to view
 * the graphs. The corresponding function is now called.
 */

void stat_ok(Widget w_id, XtPointer client_data, XmPushButtonCallbackStruct*cbs)
{
    XtUnmanageChild(client_data);
    stat_button(rowcol);
}

/*
 * * * * *
 *   Filename : file_select.c
 * * * * *
 *   Programmed by : N.Sunil Chakravarthy.
 * * * * *
 *   Last updated on : 08.17.94
 * * * * *
 */

This file contains the code for getting the file name from the user
while checking in a file. This dialog is called after pressing libin button.
*/

#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/PushButton.h>
#include <Xm/FileSB.h>
#include <Xm/DialogS.h>
#include <sys/stat.h>
#include <X11/Xos.h>
#include <Xm/DrawingA.h>
#include <Xm/DrawnB.h>
#include <Xm/Text.h>
#include <Xm/LabelG.h>
#include <Xm/RowColumn.h>
#include <Xm/Form.h>
#include <Xm/ToggleBG.h>
#include <Xm/SeparatoG.h>
#include <Xm/PushBG.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <ctype.h>
#include <errno.h>

void do_search(), new_file_cb(), file_exit(), errordialog(), destroy();
void help();
void load_help();

```

```

extern rowcol;
extern toplevel;
extern XmStringCharSet charset;
char file_name[100];
char dir[100];
char help_data[50][5000];
extern struct file_node {
    char name[20];
    char rcsfile[80];
    char rcsno[2000];
    char author[20];
    char email[20];
    char function[100];
    char method[100];
    char implementation[1000];
    float saved;
    int outno;
    int status;
    int filesize;
    struct file_node *left;
    struct file_node *right;
    struct file_node *parent;
};
extern struct file_node *point, *head;
extern char lib_d[100];

extern map();

/* This function creates a standard file selection dialog of Motif.*/

extern void interactive();
Widget file_dialog;
void file_sel(Widget w)
{
    Arg args[10];

    XtSetArg(args[0], XmNfileSearchProc, do_search);
    XtSetArg(args[1], XmNautoUnmanage, False);
    XtSetArg(args[2], XmNdefaultPosition, False);

    /* This dialog is "application modal" type. Without interacting with
     * this dialog, the user is not allowed to interact with any other
     * portion of the user interface.
     */

    XtSetArg(args[3], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);
    file_dialog = XmCreateFileSelectionDialog(w, "Files", args, 4);

    /* Position this dialog at (100,40) coordinate.*/
    XtAddCallback(file_dialog, XmNmapCallback, map, 100040);

    /* Link the necessary help callback function. */
    XtAddCallback(file_dialog, XmNhelpCallback, help, 0);

    /* Desensitize the help button. Help is only provided on pressing the F1 key.*/
    XtSetSensitive(XmFileSelectionBoxGetChild(file_dialog, XmDIALOG_HELP_BUTTON),
        False);

    /* If user presses the OK button, call new_file_cb(). */
    XtAddCallback(file_dialog, XmNokCallback, new_file_cb, NULL);

    /* If user presses Cancel button, exit program. */
    XtAddCallback(file_dialog, XmNcancelCallback, file_exit, NULL);
    XtManageChild(file_dialog);
}

/* Routine to determine if a file is accessible, a directory,
 * or writable. Return -1 on all errors or if the file is not
 * writable. Return 0 if it is a directory or 1 if it is a plain
 * writable file.
 */

```

```

int
is_writable(file)
char *file;
{
    struct stat s_buf;

    /* If file can't be accessed (via stat()) return. */
    if (stat(file, &s_buf) == -1)
        return -1;
    else if ((s_buf.st_mode & S_IFMT) == S_IFDIR)
        return 0; /* a directory */
    else if (!(s_buf.st_mode & S_IFREG) || access(file, W_OK) == -1)
        /* Not a normal file or it is not writable */
        return -1;

    /* Legitimate file */
    return 1;
}

/* A new file was selected -- check to see if it's readable and not
 * a directory. If it's not readable, report an error. If it's a
 * directory, scan it just as though the user had typed it in the mask
 * Text field and selected "Search".
 */

void
new_file_cb(w, client_data, cbs)
Widget w;
XmFileSelectionBoxCallbackStruct *cbs;
{
    char *file, name[100];
    int i, k;

    if (!XmStringGetLtoR(cbs->value, charset, &file))
        return;
    if (*file != '/') {
        /* If it's not a directory, determine the full pathname
         * of the selection by concatenating it to the "dir" part.
         */
        char *dir, *newfile;
        if (XmStringGetLtoR(cbs->dir, charset, &dir)) {
            sprintf(newfile, "%s/%s", dir, file);
            XtFree(dir);
            file = newfile;
        }
    }
    switch (is_writable(file)) {
    case 1 :
        sprintf(name, "%s", file);
        i = 0;
        while (name[i] != '\0')
            i++;
        while (name[i] != '/' && i != 0)
            i--;
        strncpy(dir, name, i + 1);
        k = 0;
        i++;
        memset(file_name, '\0', 100);
        while (name[i] != '\0')
            file_name[k++] = name[i++];

        /* Check to see if a file with the same name is present in the library. */
        point = NULL;
        traverse(head, file_name);
        XtUnmanageChild(file_dialog);
        if (point) {
            /* File with same name present give error message and allow the

```

```

        user to choose again. */
        errordialog("file with same name present in library, checkin after
                    renaming the file....");
    } else
        interactive(rowcol);
    break;
case 0 :
    {
        /* A directory was selected, scan it. */

        XmString str = XmStringCreateSimple(file);
        XmFileSelectionDoSearch(w, str);
        XmStringFree(str);
    }
    break;
case -1 :

    /* A system error on this file. */

    XtUnmanageChild(file_dialog);
    errordialog("FILE IS NOT ACCESSIBLE OR ABSENT...");
}
XtFree(file);
}

/* do_search() -- scan a directory and report only those files that
 * are writable. Here, we let the shell expand the (possible)
 * wildcards and return a directory listing by using popen().
 */

void
do_search(fs, cbs)
Widget fs; /* file selection box widget */
XmFileSelectionBoxCallbackStruct *cbs;
{
    char *mask, buf[1000], *p;
    XmString names[256]; /* maximum of 256 files in dir */
    int i = 0;
    FILE *pp, *popen();

    if (!XmStringGetLtoR(cbs->mask, charset, &mask))
        return; /* can't do anything */

    sprintf(buf, "/bin/ls %s", mask);
    XtFree(mask);

    /* Let the shell read the directory and expand the filenames. */
    if (!(pp = popen(buf, "r")))
        return;

    /* Read output from popen() -- this will be the list of files. */
    while (fgets(buf, sizeof buf, pp)) {
        if (p = index(buf, '\n'))
            *p = 0;

        /* Only list files that are writable and are not directories. */
        if (is_writable(buf) == 1 && (names[i] = XmStringCreateSimple(buf)))
            i++;
    }
    pclose(pp);
    if (i) {
        XtVaSetValues(fs, XmNfileListItems, names, XmNfileListItemCount,
                    i, XmNdirSpec, names[0], XmNlistUpdated, True,
                    NULL);
        while (i > 0)
            XmStringFree(names[--i]);
    } else
        XtVaSetValues(fs, XmNfileListItems, NULL, XmNfileListItemCount,
                    0, XmNlistUpdated, True, NULL);
}

/* The exit button is pressed in file dialog. Pop down the file_selection dialog and

```

```

* return to screen 2. Check-in process is aborted.
*/

void
file_exit(w, client_data, cbs)
Widget w;
XmFileSelectionBoxCallbackStruct *cbs;
{
    XtUnmanageChild(w);
}

/* This function is used throughout the application to pop up error messages.
* The error message passed as an argument, is displayed in a information dialog.
*/

void errordialog(char str[1000])
{
    Widget dialog;
    XmString text;
    dialog = XmCreateInformationDialog(toplevel, "MESSAGE", NULL, 0);
    text = XmStringCreateSimple(str);
    XtVaSetValues(dialog, XmNmessageString, text, XmNdialogStyle,
                  XmDIALOG_FULL_APPLICATION_MODAL, NULL);
    XtSetSensitive(XmMessageBoxGetChild(dialog, XmDIALOG_HELP_BUTTON), False);
    XmStringFree(text);
    XtManageChild(dialog);
}

/* This function is used as a callback to pop down the dialog passed to it as an argument.
* This callback is usually linked to the "CANCEL" button in all the dialogs.
*/

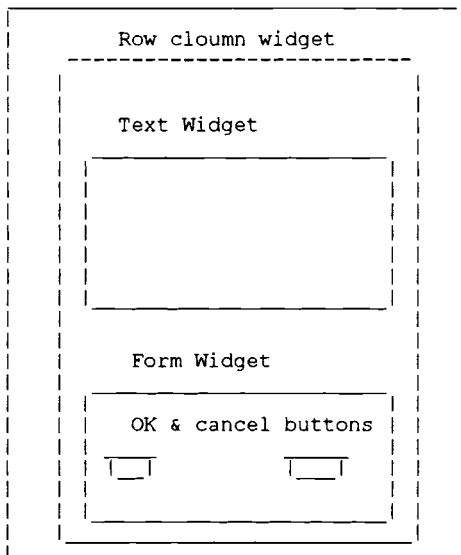
void destroy(Widget w_id, XtPointer client_data, XmPushButtonCallbackStruct *cbs)
{
    XtDestroyWidget(client_data);
}

/* This function is used to pop up a help dialog when the F1 key is pressed.
* The client data identifies where the F1 key was pressed and by indexing into the
* help_data array the corresponding help text is obtained which is displayed
* in a text widget.
*/

/*
The structure of this dialog box is as follows.

```

Bulletin Board Widget




```

*/

void help(Widget w_id, int client_data, XmAnyCallbackStruct *cbs)
{
    Widget dialog, text_w1, rowcol2, rowcol3, pb1, pb2;
    XmString text, one, two;
    Arg ags[10];
    char temp[2000];

    XtSetArg(ags[0], XmNautoUnmanage, True);
    XtSetArg(ags[1], XmNdefaultPosition, False);
    XtSetArg(ags[2], XmNallowShellResize, True);
    dialog = XmCreateBulletinBoardDialog(w_id, "HELP", ags, 3);

    /* Position the dialog at (150,100) coordinate. */

    XtAddCallback(dialog, XmNmapCallback, map, 150100);

    rowcol2 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, dialog,
                               XmNnumColumns, 1, XmNorientation, XmVERTICAL, NULL);
    XtSetArg(ags[0], XmNrows, 20);
    XtSetArg(ags[1], XmNcolumns, 60);
    XtSetArg(ags[2], XmNeditable, False);
    XtSetArg(ags[3], XmNeditMode, XmMULTI_LINE_EDIT);
    XtSetArg(ags[4], XmNwordWrap, True);
    if (client_data != 10)
        XtSetArg(ags[5], XmNvalue, help_data[client_data]);
    else {
        sprintf(temp, "%s%s", help_data[10], help_data[11]);
        XtSetArg(ags[5], XmNvalue, temp);
    }
    XtSetArg(ags[6], XmNscrollHorizontal, False);
    text_w1 = XmCreateScrolledText(rowcol2, "text", ags, 7);
    rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcol2, NULL);
    one = XmStringCreateSimple("OK");
    two = XmStringCreateSimple("CANCEL");
    pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
                                   XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
                                   XmNleftOffset, 5, XmNheight, 25, NULL);

    pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
                                   XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
                                   XmNrightOffset, 5, XmNheight, 25, NULL);
    XtAddCallback(pb1, XmNactivateCallback, destroy, dialog);
    XtAddCallback(pb2, XmNactivateCallback, destroy, dialog);

    XtFree(one);
    XtFree(two);
    XtManageChild(text_w1);
    XtManageChild(rowcol2);
    XtManageChild(rowcol3);
    XtManageChild(dialog);
}

/* This function is used to load the help text from the help.dat file into the
 * character array help_data. This function is called at the beginning of the
 * application while creating screen1.
 */

void load_help()
{
    FILE * fp;
    char str[100], s[5000];
    int i;
    sprintf(str, "%shelp.dat", lib_d);
    fp = fopen(str, "r");
    if (!fp) {
        errorDialog("Unable to load help file");
    } else {
        i = 0;
        while (!feof(fp)) {
            fgets(str, 80, fp);

            /* When "*" is seen in the first position (while loading lines from
             files), one topic is over. Load the next topic after incrementing

```

```

        the array index.
    */

    while (!feof(fp) && (str[0] != '*')) {
        strcat(s, str);
        fgets(str, 80, fp);
    }
    strcpy(help_data[i], s);
    memset(s, '\0', 5000);
    i++;
}
}

/*
    * * * * *
    *   Filename : Description.c
    *
    *   Programmed by : N.Sunil Chakravarthy.
    *
    *   Last updated on : 08.17.94
    * * * * *
*/

```

Once the name of the file is selected the description of the file is accepted from the user.
This file contains the code for the description dialog.

```

/*
#include <Xm/DialogS.h>
#include <Xm/TextF.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/Form.h>
#include <Xm/SelectioB.h>
#include <Xm/PushB.h>
#include <Xm/LabelG.h>
int  description_ok();
void Destroy();
void modal_error_dialog();
extern destroy();
extern help();
extern rowcol;

extern struct file_node {
    char  name[20];
    char  rcsfile[80];
    char  rcsno[2000];
    char  author[20];
    char  email[20];
    char  function[100];
    char  method[100];
    char  implementation[1000];
    float saved;
    int   outno;
    int   status;
    int   filesize;
    struct file_node *left;
    struct file_node *right;
    struct file_node *parent;
};

struct file_node *new_file;
extern Widget toplevel;

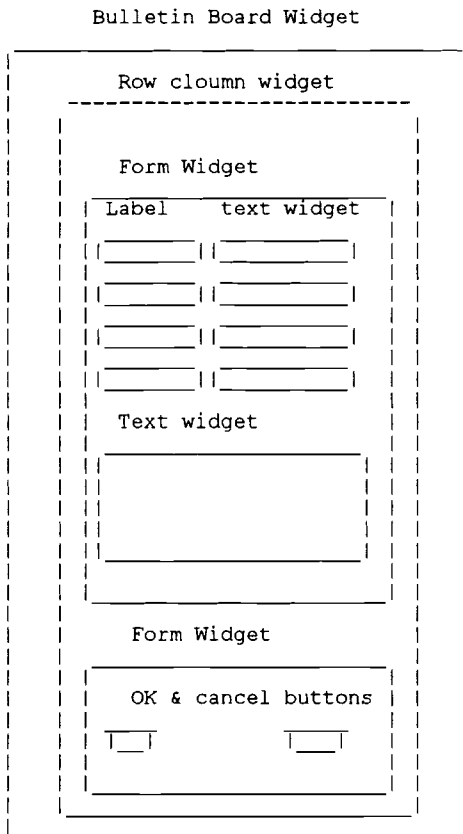
extern void map();
void desc_map();

Widget desc_dialog, name2, addr2, text_w1, func2, method2;

/* This function is called to display the description dialog after the file
 * name has been accepted.
 */

```

/* The structure of this dialog box is as follows.



```

*/
void interactive(Widget w)
{
    Widget label, label1, rowcol2, rowcol3, pb1, pb2, form1, name1, addr1,
        func1, method1;
    Arg ags[10];
    XmString one, two;
    char str[80];
    int pos;
    int i = 0;
    memset(str, '\0', 80);
    XtSetArg(ags[i], XmNautoUnmanage, False);
    i++;
    XtSetArg(ags[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(ags[i], XmNallowShellResize, True);
    i++;
    XtSetArg(ags[i], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);
    desc_dialog = XmCreateBulletinBoardDialog(toplevel, "DESCRIPTION",
        ags, 4);

    /* Position the dialog at 200,100. */

    XtAddCallback(desc_dialog, XmNmapCallback, desc_map, 200100);
    XtAddCallback(desc_dialog, XmNhelpCallback, help, 1);

    rowcol2 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, desc_dialog,
        XmNnumColumns, 1, XmNorientation, XmVERTICAL, NULL);
    form1 = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcol2, NULL);

```

```

one = XmStringCreateSimple("NAME OF THE AUTHOR");
name1 = XtVaCreateWidget("label", xmLabelWidgetClass, form1, XmNlabelString,
    one, XmNleftAttachment, XmATTACH_FORM, XmNleftOffset, 5, XmNheight,
    25, NULL);
XtSetArg(ags[0], XmNrows, 1);
XtSetArg(ags[1], XmNcolumns, 20);
XtSetArg(ags[2], XmNleftAttachment, XmATTACH_WIDGET);
XtSetArg(ags[3], XmNleftWidget, name1);
XtSetArg(ags[4], XmNtopAttachment, XmATTACH_FORM);
XtSetArg(ags[5], XmNtopOffset, 5);
XtSetArg(ags[6], XmNrightAttachment, XmATTACH_FORM);
XtSetArg(ags[7], XmNvalue, str);
name2 = XmCreateText(form1, "text", ags, 8);
XtAddCallback(name2, XmNactivateCallback, XmProcessTraversal,
    XmTRAVERSE_NEXT_TAB_GROUP);
XtFree(one);
one = XmStringCreateSimple("EMAIL ADDRESS");
addr1 = XtVaCreateWidget("label", xmLabelWidgetClass, form1, XmNlabelString,
    one, XmNleftAttachment, XmATTACH_FORM, XmNleftOffset, 5, XmNtopAttachment,
    XmATTACH_WIDGET, XmNtopWidget, name1, XmNtopOffset, 25, XmNheight,
    25, NULL);
XtSetArg(ags[0], XmNrows, 1);
XtSetArg(ags[1], XmNcolumns, 20);
XtSetArg(ags[2], XmNleftAttachment, XmATTACH_WIDGET);
XtSetArg(ags[3], XmNleftWidget, addr1);
XtSetArg(ags[4], XmNtopAttachment, XmATTACH_WIDGET);
XtSetArg(ags[5], XmNtopWidget, name2);
XtSetArg(ags[6], XmNtopOffset, 10);
XtSetArg(ags[7], XmNrightAttachment, XmATTACH_FORM);
XtSetArg(ags[8], XmNvalue, str);
addr2 = XmCreateText(form1, "text", ags, 9);
XtAddCallback(addr2, XmNactivateCallback, XmProcessTraversal,
    XmTRAVERSE_NEXT_TAB_GROUP);
XtFree(one);
one = XmStringCreateSimple("FUNCTION");
func1 = XtVaCreateWidget("label", xmLabelWidgetClass, form1, XmNlabelString,
    one, XmNleftAttachment, XmATTACH_FORM, XmNleftOffset, 5, XmNtopAttachment,
    XmATTACH_WIDGET, XmNtopWidget, addr1, XmNtopOffset, 25, XmNheight,
    15, NULL);
XtSetArg(ags[0], XmNrows, 1);
XtSetArg(ags[1], XmNcolumns, 20);
XtSetArg(ags[2], XmNleftAttachment, XmATTACH_WIDGET);
XtSetArg(ags[3], XmNleftWidget, func1);
XtSetArg(ags[4], XmNtopAttachment, XmATTACH_WIDGET);
XtSetArg(ags[5], XmNtopWidget, addr2);
XtSetArg(ags[6], XmNtopOffset, 10);
XtSetArg(ags[7], XmNrightAttachment, XmATTACH_FORM);
XtSetArg(ags[8], XmNvalue, str);
func2 = XmCreateText(form1, "text", ags, 9);
XtAddCallback(func2, XmNactivateCallback, XmProcessTraversal,
    XmTRAVERSE_NEXT_TAB_GROUP);
XtFree(one);
one = XmStringCreateSimple("METHOD");
method1 = XtVaCreateWidget("label", xmLabelWidgetClass, form1, XmNlabelString,
    one, XmNleftAttachment, XmATTACH_FORM, XmNleftOffset, 5, XmNtopAttachment,
    XmATTACH_WIDGET, XmNtopWidget, func1, XmNtopOffset, 15, XmNheight,
    25, NULL);
XtSetArg(ags[0], XmNrows, 1);
XtSetArg(ags[1], XmNcolumns, 20);
XtSetArg(ags[2], XmNleftAttachment, XmATTACH_WIDGET);
XtSetArg(ags[3], XmNleftWidget, method1);
XtSetArg(ags[4], XmNtopAttachment, XmATTACH_WIDGET);
XtSetArg(ags[5], XmNtopWidget, func2);
XtSetArg(ags[6], XmNtopOffset, 10);
XtSetArg(ags[7], XmNrightAttachment, XmATTACH_FORM);
XtSetArg(ags[8], XmNvalue, str);
method2 = XmCreateText(form1, "text", ags, 9);
XtAddCallback(method2, XmNactivateCallback, XmProcessTraversal,
    XmTRAVERSE_NEXT_TAB_GROUP);
XtFree(one);
one = XmStringCreateSimple("IMPLEMENTATION DETAILS:");
label1 = XtVaCreateWidget("label", xmLabelWidgetClass, rowcol2,
    XmNlabelString, one, NULL);
XtSetArg(ags[0], XmNrows, 5);
XtSetArg(ags[1], XmNcolumns, 10);

```

```

XtSetArg(ags[2], XmNeditable, True);
XtSetArg(ags[3], XmNeditMode, XmMULTI_LINE_EDIT);
XtSetArg(ags[4], XmNwordWrap, True);
XtSetArg(ags[5], XmNvalue, str);
text_w1 = XmCreateScrolledText(rowcol2, "text", ags, 6);
rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcol2, NULL);
one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
                               XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
                               XmNleftOffset, 5, XmNheight, 25, NULL);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
                               XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
                               XmNrightOffset, 5, XmNheight, 25, NULL);

XtAddCallback(pb1, XmNactivateCallback, description_ok, text_w1);
XtAddCallback(pb2, XmNactivateCallback, destroy, desc_dialog);

XtFree(one);
XtFree(two);

XtManageChild(text_w1);
XtManageChild(labe11);
XtManageChild(name1);
XtManageChild(name2);
XtManageChild(addr1);
XtManageChild(addr2);
XtManageChild(func1);
XtManageChild(func2);
XtManageChild(method1);
XtManageChild(method2);
XtManageChild(form1);
XtManageChild(rowcol2);
XtManageChild(rowcol3);
XtManageChild(desc_dialog);
}

/* This function is called as soon as the user pushes the OK button in the
 * description dialog. This function is used to check if any information
 * is provided in the name, e-mail and the function portion of the description.
 * If provided, the name, e-mail, function, method and implementation details
 * are recorded in the file structure.
 */

int description_ok(Widget w_id, XtPointer w, XmAnyCallbackStruct *cbs)
{
    char *text, *text1, *text2, *text3, *text4;
    text = XmTextGetString(name2);
    text1 = XmTextGetString(addr2);
    text2 = XmTextGetString(func2);
    text3 = XmTextGetString(method2);
    text4 = XmTextGetString(text_w1);
    if (strlen(text) < 1 || strlen(text1) < 1 || strlen(text2) < 1) {

        /* Give error message because name, e-mail, or function description
         * is missing.
         */

        modal_error_dialog(desc_dialog, "name,e-mail address, and function
            description must be provided");
        XtManageChild(desc_dialog);
        return;
    }

    /* Check the lengths of various descriptions. */

    if (strlen(text) > 19) {
        error_dialog("NAME IS TOO LONG ....");
        return;
    }

    if (strlen(text1) > 40) {
        error_dialog("E_MAIL ADDRESS IS TOO LONG.....");
        return;
    }
}

```

```

if (strlen(text2) > 99) {
    errordialog("FUNCTION IS TOO LONG ....");
    return;
}
if (strlen(text3) > 99) {
    errordialog("METHOD IS TOO LONG ....");
    return;
}
if (strlen(text4) > 999) {
    errordialog("IMPLEMENTATION DETAILS IS TOO LONG .....");
    return;
}

/* Create a new node to be inserted in the tree and record the above
 * descriptions in it.
 */

new_file = (struct file_node *)malloc(sizeof(struct file_node ));
if (!new_file) {
    errordialog("UNABLE TO MALLOC .....");
    return;
}
strcpy(new_file->author, text);
strcpy(new_file->email, text1);
strcpy(new_file->function, text2);
strcpy(new_file->method, text3);
strcpy(new_file->implementation, text4);
new_file->left = NULL;
new_file->right = NULL;
new_file->parent = NULL;

XtFree(text);
XtFree(text1);
XtFree(text2);
XtFree(text3);
XtFree(text4);

/* Call the next function. */

XtUnmanageChild(desc_dialog);
lib(rowcol);
}

/* This function is used to destroy a widget. This is used as callback function for the
cancel button.*/

void Destroy(Widget w)
{
    XtDestroyWidget(w);
}

/* This function is used to position a dialog within the toplevel shell. */

void desc_map(Widget tw, int w, XmAnyCallbackStruct *cbs)
{
    int    xpos = 0, ypos = 0, x1 = 0, y1 = 0;

    /* Decode the coordinates sent as one int. */

    x1 = w / 1000;
    y1 = w - x1 * 1000;

    /* Get the coordinates of the toplevel shell. */

    XtVaGetValues(toplevel, XmNx, &xpos, XmNy, &ypos, NULL);

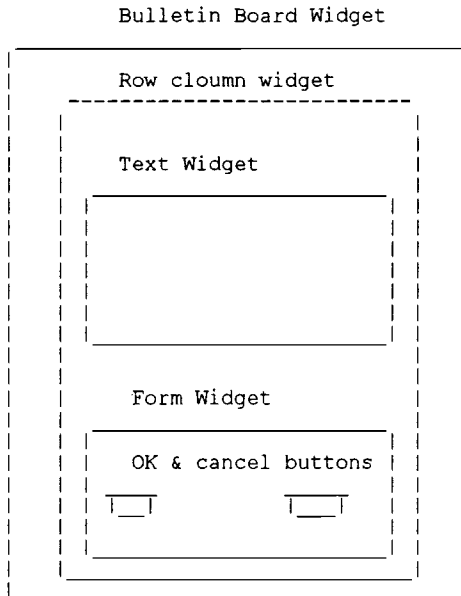
    /* Position the dialog in the dialog shell. */

    XtVaSetValues(tw, XmNx, xpos + x1, XmNy, ypos + y1, NULL);
}

/* This function pops up the error dialog with the required message.
 * First there were two errordialogs, one with normal style and this
 * dialog with application modal style. Later, the normal one was
 * converted to full application modal. This dialog persists because
 * of its geometry.
 */

```

/* The structure of this dialog box is as follows.



*/

```

void modal_error_dialog(Widget wid, char str[1000])
{
    Widget dialog, text_w1, rowcol2, rowcol3, pb1, pb2;
    XmString text, one, two;
    Arg ags[10];
    char temp[2000];

    XtSetArg(ags[0], XmNautoUnmanage, True);
    XtSetArg(ags[1], XmNdefaultPosition, False);
    XtSetArg(ags[2], XmNallowShellResize, True);
    dialog = XmCreateBulletinBoardDialog(wid, "ERROR", ags, 3);
    XtAddCallback(dialog, XmNmapCallback, map, 150100);

    rowcol2 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, dialog,
        XmNnumColumns, 1, XmNorientation, XmVERTICAL, NULL);
    XtSetArg(ags[0], XmNrows, 5);
    XtSetArg(ags[1], XmNcolumns, 60);
    XtSetArg(ags[2], XmNeditable, False);
    XtSetArg(ags[3], XmNeditMode, XmMULTI_LINE_EDIT);
    XtSetArg(ags[4], XmNwordWrap, True);
    XtSetArg(ags[5], XmNvalue, str);
    XtSetArg(ags[6], XmNscrollHorizontal, False);
    text_w1 = XmCreateScrolledText(rowcol2, "text", ags, 7);
    rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcol2,
        NULL);
    one = XmStringCreateSimple("OK");
    two = XmStringCreateSimple("CANCEL");
    pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM, XmNleftOffset,
        5, XmNheight, 25, NULL);

    pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
        XmNrightOffset, 5, XmNheight, 25, NULL);
    XtAddCallback(pb1, XmNactivateCallback, destroy, dialog);
    XtAddCallback(pb2, XmNactivateCallback, destroy, dialog);

    XtFree(one);
    XtFree(two);
}
  
```

```

XtManageChild(text_w1);
XtManageChild(rowcol2);
XtManageChild(rowcol3);
XtManageChild(dialog);
}

/*
* * * * *
*   Filename : Position.c
* * * * *
*   Programmed by : N.Sunil Chakravarthy.
* * * * *
*   Last updated on : 08.17.94
* * * * *
*/

This file contains the code to pop up the dialog asking the user to choose
either "Position selected by user" or "Position selected by system ".
*/

#include <Xm/DialogS.h>
#include <Xm/TextF.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/Form.h>
#include <Xm/SelectioB.h>
#include <Xm/PushB.h>
#include <Xm/LabelG.h>
#include <Xm/Label.h>
#include <Xm/Scale.h>
# include <stdio.h>
# include <X11/cursorfont.h>

void user_select(), sel_by_system(), check_in(), Scale_widget();
void toggle();
void ForceUpdate();
extern rowcol;

extern void destroy();
extern void Destroy();
extern help();
extern void list_display();
extern void system_select();
extern void add_node();
extern void write_exp();
extern void map();
extern struct file_node {
    char    name[20];
    char    rcsfile[80];
    char    rcsno[2000];
    char    author[20];
    char    email[20];
    char    function[100];
    char    method[100];
    char    implementation[1000];
    float   saved;
    int     outno;
    int     status;
    int     filesize;
    struct file_node *left;
    struct file_node *right;
    struct file_node *parent;
};

extern struct file_node *new_file, *point, *head;

extern char    dir[100], file_name[100], src_d[100], lib_d[100];
extern char    *passwd;

extern int     gl_test;
extern int     sys_adm;
extern char    filename[80];
extern Widget  toplevel;
extern int     tcount;

```



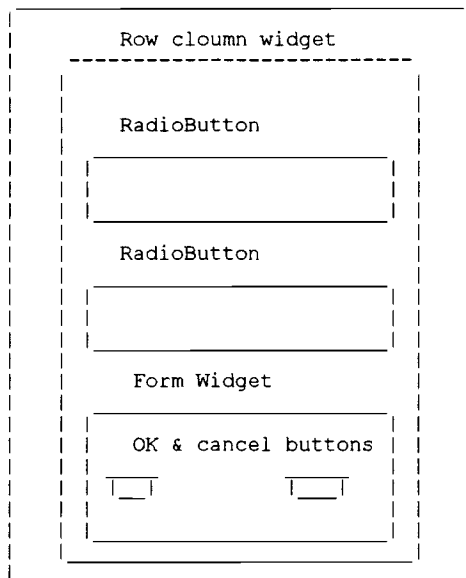
```
extern void countfiles();

int mode, num;

/* This function is called after the description is provided by the user.
 * This function allows the user to choose either "Position selected
 * by user" or "position selected by system ".
 */

/* The structure of this dialog box is as follows.
```

Bulletin Board Widget



```
*/

Widget indialog, scaledialog, scale;
void lib(Widget w)
{
    Widget pb1, cin_R, cin_N, cin_S, cin_T, cin_M, rowcol1, pb2, rowcol3,
        rad;
    XmString one, two, three, four, five;
    Arg args[10];
    int i;
    i = 0;
    XtSetArg(args[i], XmNautoUnmanage, False);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNwidth, 100);
    i++;
    XtSetArg(args[i], XmNheight, 100);
    i++;
    XtSetArg(args[i], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);

    indialog = XmCreateBulletinBoardDialog(toplevel, "LIBIN", args, 6);

    /* Position the dialog at 200,100. */

    XtAddCallback(indialog, XmNmapCallback, map, 200100);
    XtAddCallback(indialog, XmNhelpCallback, help, 2);

    mode = 0;
    rowcol1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, indialog,
        XmNnumColumns, 1, XmNpacking, XmPACK_COLUMN, XmNorientation,
        XmVERTICAL, NULL);
    one = XmStringCreateSimple("Position Suggested By User");
```

```

two = XmStringCreateSimple("Position Suggested By System");
rad = XmVaCreateSimpleRadioBox(rowcol1, "radio box", 0, toggle, XmVaRADIOBUTTON,
    one, 0, NULL, NULL, XmVaRADIOBUTTON, two, 0, NULL, NULL, NULL);

rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcol1, NULL);

one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
    XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
    XmNbottomAttachment, XmATTACH_FORM, XmNleftOffset, 5,
    XmNheight, 25, NULL);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
    XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
    XmNbottomAttachment, XmATTACH_FORM, XmNrightOffset,
    5, XmNheight, 25, NULL);

XtAddCallback(pb1, XmNactivateCallback, check_in, indialog);
XtAddCallback(pb2, XmNactivateCallback, destroy, indialog);
XmStringFree(one);
XmStringFree(two);
XtManageChild(rowcol1);
XtManageChild(rowcol3);
XtManageChild(rad);
XtManageChild(indialog);
}

/* This function is used to toggle the radiobutton's pixmap to indicate if it
 * is pushed or not.
 */

void toggle(Widget w_id, int client_data, XmToggleButtonCallbackStruct *cbs)
{
    if (cbs->set == False)
        return;
    mode = client_data;
}

/* "Position selected by system" was pushed, record it. */

void sel_by_system(Widget w_id, XtPointer client_data, XmPushButtonCallbackStruct *cbs)
{
    mode = 1;
}

/* "Position selected by user" was pushed, record it. */

void user_select(Widget w_id, XtPointer client_data, XmPushButtonCallbackStruct *cbs)
{
    mode = 0;
}

/* This function is called when the user pushes the OK button in the dialog.
 * The variable "mode" records the choice of the user.
 * If the choice is selection by system, call the system-select function to locate the
 * best parent for the file. After this, checkin the file to this parent. This
 * completes the Libin process.
 * If the choice is the position selected by user, call function list display to
 * display the list to allow the user to select a suitable parent.
 */

void check_in(Widget w_id, XtPointer client_data, XmPushButtonCallbackStruct *cbs)
{
    int    i, j;
    char  s[800], r[80], str[100], *g;
    FILE * fp;
    Cursor cursor;
    XtUnmanageChild(indialog);
    if (mode) {

        /* SYSTEM SELECT ROUTINE*/

        gl_test = 100000000;
    }
}

```

```

memset(filename, '\0', 80);
strcpy(src_d, dir);

/* Change the shape of the cursor to show working status. */

cursor = XCreateFontCursor(XtDisplay(toplevel), 26);
XDefineCursor(XtDisplay(toplevel), XtWindow(toplevel), cursor);
XSync(XtDisplay(toplevel), False);
tcount = 0;
countfiles(head->left);

/* Pop up the working dialog with a scale widget. */

Scale_widget(tcount);
num = 0;

/* Call the system select routine. */

system_select(head->left, file_name);

/* Restore the original shape of the cursor. */

cursor = XCreateFontCursor(XtDisplay(toplevel), 58);
XDefineCursor(XtDisplay(toplevel), XtWindow(toplevel), cursor);
XmScaleSetValue(scale, 100);
sleep(1);

/* Pop down the working dialog. */

XtDestroyWidget(scaledialog);

/* Initiate the check-in process. */

if (filename[0] == '\0') {
    strcpy(filename, "head");
}
strcpy(s, file_name);
q = strtok(s, ".");
if (q)
    sprintf(r, "rcs%s", q);
else
    sprintf(r, "rcs%s", file_name);
strcpy(new_file->rcsno, r);

strcpy(new_file->name, file_name);

/* Only if the user is the library administrator is the file actually
 * checked into the system.
 * Otherwise, it is recorded as an experimental file in Experimental.dat
 */

if (sys_adm) {

    /* library administrator, so actually check-in the file. */

    new_file->status = 1;
    add_node(new_file, filename);
} else {

    /* User, so add to the Experimental.dat file. */

    sprintf(str, "cp %s%s %s%s", src_d, file_name, lib_d,
            file_name);
    system(str);
    memset(str, '\0', 80);
    sprintf(str, "chmod 777 %s%s", lib_d, file_name);
    system(str);
    write_exp(new_file, filename);
    new_file->status = 0;
    add_exp(new_file, filename);
}
} else {

    /* Position selected by user routine. */

    list_display(rowcol);
}
}

```

```

/* This function pops up a working dialog with a scale widget showing what
 * percentage is complete.
 */

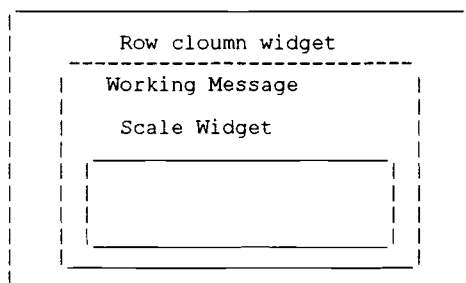
```

```

/*
The structure of this dialog box is as follows.

```

Bulletin Board Widget



```

*/
void Scale_widget(int count)
{
    Widget rowcoll, label, labell;
    XmString one, two, three, four, five;
    Arg args[10];
    char c[80];
    int i;
    i = 0;
    XtSetArg(args[i], XmNautoUnmanage, True);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNwidth, 100);
    i++;
    XtSetArg(args[i], XmNheight, 100);
    i++;
    XtSetArg(args[i], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);
    i++;
    scaledialog = XmCreateBulletinBoardDialog(toplevel, "WORKING", args, 6);
    XtAddCallback(scaledialog, XmNmapCallback, map, 150125);

    rowcoll = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, scaledialog, NULL);
    one = XmStringCreateSimple(" WORKING PLEASE WAIT");
    label = XtVaCreateManagedWidget("label", xmLabelWidgetClass, rowcoll,
        XmNlabelString, one, NULL);
    XtFree(one);
    one = XmStringCreateSimple("Percentage Done");
    labell = XtVaCreateManagedWidget("label", xmLabelWidgetClass, rowcoll,
        XmNlabelString, one, NULL);
    XtFree(one);

    scale = XtVaCreateManagedWidget("files", xmScaleWidgetClass, rowcoll,
        XmNmaximum, 100, XmNminimum, 0, XmNvalue, 0, XmNscaleWidth, 300,
        XmNshowValue, True, XmNsensitive, False, XmNorientation, XmHORIZONTAL,
        XmNprocessingDirection, XmMAX_ON_RIGHT, NULL);

    XtManageChild(rowcoll);
    XtManageChild(scaledialog);
    ForceUpdate(scaledialog);
}

```

```

/* This function is used to force the update of a dialog onto the screen. This is
 * needed in the working dialog because it will not be mapped onto the screen as
 * the processor starts the long procedure before the dialog has a chance to be
 * mapped. This function makes the processor to wait till the dialog is mapped.
 */

```

```

void ForceUpdate(Widget w)
{

```

```

Widget diashell, topshell;
Window diawindow, topwindow;
XtAppContext cxt = XtWidgetToApplicationContext(w);
Display * dpy;
XWindowAttributes xwa;
XEvent event;
for (diashell = w; !XtIsShell(diashell); diashell = XtParent(diashell))
;
for (topshell = diashell; !XtIsTopLevelShell(topshell); topshell
    = XtParent(topshell))
;
if (XtIsRealized(diashell) && XtIsRealized(topshell)) {
    dpy = XtDisplay(topshell);
    diawindow = XtWindow(diashell);
    topwindow = XtWindow(topshell);

    while (XGetWindowAttributes(dpy, diawindow, &xwa) && xwa.map_state !=
        IsViewable) {
        if (XGetWindowAttributes(dpy, topwindow, &xwa) &&
            xwa.map_state != IsViewable)
            break;
        XtAppNextEvent(cxt, &event);
        XtDispatchEvent(&event);
    }
    XmUpdateDisplay(topshell);
}

/*
* * * * *
*   Filename : User_select.c
*
*   Programmed by : N. Sunil Chakravarthy.*
*
*   Last updated on : 08.17.94
* * * * *

```

This file has the code to pop up the dialogs in the last stage of the user selected position in libin and the last stage of file selected by name in libout.

```

*/

#include <Xm/List.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/DialogS.h>
#include <Xm/LabelG.h>
#include <Xm/Form.h>
#include <Xm/PushButton.h>

extern XmStringCharSet charset;
extern rowcol;
extern destroy();
extern help();
extern errordialog();
extern checkout();
extern Widget show();
extern void map();
extern traverse();

/* Defined and explained in load.c */

extern struct file_node {
    char    name[20];
    char    rcsfile[80];
    char    rcsno[2000];
    char    author[20];
    char    email[20];
    char    function[100];
    char    method[100];
    char    implementation[1000];
    float   saved;
    int     outno;

```

```

        int     status;
        int     filesize;
        struct file_node *left;
        struct file_node *right;
        struct file_node *parent;
};

extern struct file_node *new_file, *point, *head, *child, *cur;

/* Defined and explained in list.c */

extern struct list {
    char    name[20];
    char    filename[20];
    int     type;
    int     distance;
    int     filesize;
    struct list *left;
    struct list *right;
};

/* Defined and explained in thesaurus.c. */

extern struct word_list {
    char    word[20];
    struct word_list *right;
};

extern struct word_list *wordhead;
extern int     checkoutflag;
extern int     sys_adm;

extern struct list *listhead;
extern char    dir[100], file_name[100], src_d[100], lib_d[100];
extern maintain_list();
extern writelist();
extern load_list();
extern write_exp();
extern void add_exp();

void list_names();
void sclr();
void text_select();
void select_list();
void alternate_list();
void alternate_names();
void alternate_words();
void load_words();
void try_thesaurus();
void set_text();
void show_desc();
void select_file();
void description();
void show_description();
void final_in();
void try_display();

extern Widget    toplevel;
Widget    top, pb1, pb2, pb3, pb4, rowcol3, list_w, sb, labell, ll, pl;
char    def[1000];

/* This function is called when the user selects the option "Position selected
 * by user" in the Libin process, or when the user selects "File selected by
 * name" in the Libout process.
 * This dialog displays the names of the files and the logical links in a
 * list widget. The user is expected to select one file name as the parent of
 * the current file being checked in. Note that this same dialog is used
 * in the libout procedure to select the file name to be checked out. A boolean
 * is used to imply which of the two modes is currently active.
 * To help the user to choose the file, the following aids are provided.
 *
 * a. Typing a file name and pushing the display button
 *    displays the DAG from that file.
 *
 * b. Double-clicking on a file name in the list displays
 *    the description of the file.

```

```

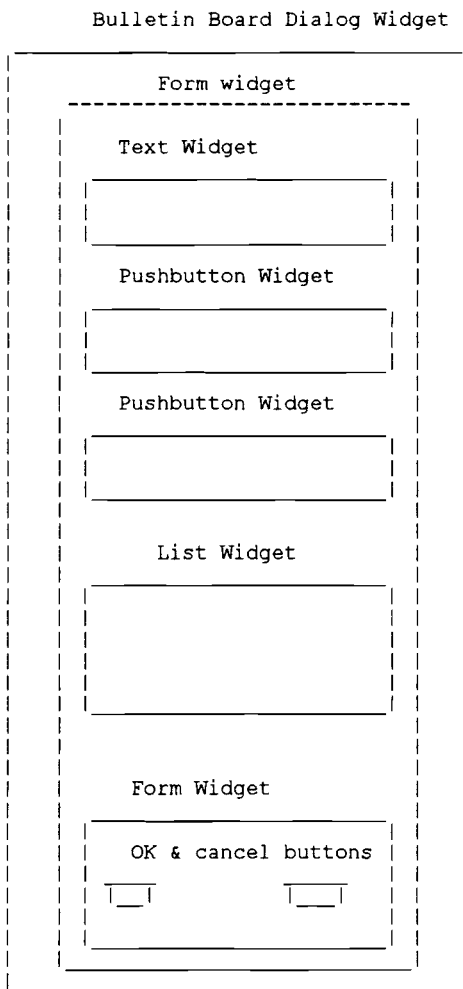
*      c. Typing a word in text widget and pushing the thesaurus
*      button invokes the thesaurus of the library which gives
*      alternate words used in the library.
*      d. Typing a link name in the text widget and pressing enter
*      causes the file names to which this link is connected to
*      be displayed.
*/

```

```

/*
The structure of this widget is.

```



```

*/

```

```

void list_display(Widget w)
{
    Arg args[10];
    int i = 0;
    XmString one, two;
    XtSetArg(args[i], XmNautoUnmanage, True);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNwidth, 400);
    i++;
    XtSetArg(args[i], XmNheight, 300);
    i++;
}

```

```

top = XmCreateBulletinBoardDialog(toplevel, "LIST", args, 5);
/* Position the dialog at 250,100. */
XtAddCallback(top, XmNmapCallback, map, 250100);
XtAddCallback(top, XmNhelpCallback, help, 3);

l1 = XtVaCreateWidget("rowcol", xmFormWidgetClass, top, NULL);

one = XmStringCreateSimple("SELECT FILE:");
labell = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
    one, XmNleftAttachment, XmATTACH_FORM, XmNleftOffset, 10, XmNtopAttachment,
    XmATTACH_FORM, XmNtopOffset, 10, XmNheight, 25, NULL);

p1 = XtVaCreateManagedWidget("text", xmTextWidgetClass, l1, XmNleftAttachment,
    XmATTACH_WIDGET, XmNleftWidget, labell, XmNleftOffset, 10, XmNtopAttachment,
    XmATTACH_FORM, XmNtopOffset, 5, XmNrightAttachment, XmATTACH_FORM,
    XmNrightOffset, 10, NULL);
XtAddCallback(p1, XmNmodifyVerifyCallback, scrl, NULL);

/* Call the text_select function when the enter key is pressed in the text widget.
*/
XtAddCallback(p1, XmNactivateCallback, text_select, NULL);

one = XmStringCreateSimple("    THESAURUS    ");
pb3 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, l1,
    XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM, XmNleftOffset,
    10, XmNtopAttachment, XmATTACH_WIDGET, XmNtopWidget, labell, XmNtopOffset,
    15, XmNrightAttachment, XmATTACH_FORM, XmNrightOffset, 10, NULL);

/* Call the try_thesaurus function when the thesaurus button is pushed. */
XtAddCallback(pb3, XmNactivateCallback, try_thesaurus, top);

one = XmStringCreateSimple("    DISPLAY    ");
pb4 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, l1,
    XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
    XmNleftOffset, 10, XmNtopAttachment, XmATTACH_WIDGET, XmNtopWidget,
    pb3, XmNtopOffset, 15, XmNrightAttachment, XmATTACH_FORM,
    XmNrightOffset, 10, NULL);

/* Call the try_display function when the display button is pushed. */
XtAddCallback(pb4, XmNactivateCallback, try_display, p1);
XtSetArg(args[0], XmNleftAttachment, XmATTACH_FORM);
XtSetArg(args[1], XmNleftOffset, 10);
XtSetArg(args[2], XmNrightAttachment, XmATTACH_FORM);
XtSetArg(args[3], XmNrightOffset, 10);
XtSetArg(args[4], XmNtopAttachment, XmATTACH_WIDGET);
XtSetArg(args[5], XmNtopWidget, pb4);
XtSetArg(args[6], XmNtopOffset, 15);

list_w = XmCreateScrolledList(l1, "list_w", args, 7);
/* Set the number of files shown in the list to 10. */
XtVaSetValues(list_w, XmNvisibleItemCount, 10, NULL);

/* Call the show_desc function for any interaction in the list widget. */
XtAddCallback(list_w, XmNdefaultActionCallback, show_desc, NULL);
rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, l1, XmNtopAttachment,
    XmATTACH_WIDGET, XmNtopWidget, list_w, XmNtopOffset, 25, XmNleftAttachment,
    XmATTACH_FORM, XmNrightAttachment, XmATTACH_FORM, NULL);
one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
    XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
    XmNleftOffset, 5, XmNheight, 25, NULL);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
    XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
    XmNrightOffset, 5, XmNheight, 25, NULL);
XtAddCallback(pb1, XmNactivateCallback, final_in, top);
XtAddCallback(pb2, XmNactivateCallback, destroy, top);
XtManageChild(rowcol3);

```



```

XtManageChild(list w);
XtManageChild(label1);

/* List w function is called to load the list of file names and logical
 * links into the list widget.
 */

list_names(list w);
XtManageChild(l1);
XtManageChild(top);
}

/* This function is called when the user pushes the display pushbutton in the
 * List dialog. This function verifies if the word typed in the text widget
 * is the name of a file in the library. If so, the display is started from
 * that file else the display is started from the head node.
 */

void try_display(Widget tw, XtPointer unused, XmTextVerifyCallbackStruct *cbs)
{
    char *text;
    Widget new;
    text = XmTextGetString(p1);

    /* Function to display. */

    /* Create the dialog for the display. */

    new = show();

    /* Check to see if the word is a valid file name. */

    point = NULL;
    traverse(head, text);
    if (point) {

        /* Yes it is a valid file name. */

        child = point->left;
        cur = point;
        treel(new);
    } else {

        /* No, it is not a valid file name, start the display from head */

        point = head;
        child = point->left;
        cur = point;
        treel(new);
    }
}

/* This function is called when the user types any character in the text
 * widget. This function is used to position the list in such a way that
 * words beginning with the string in the text widget are displayed at the
 * top of the list. The method employed is as follows. As soon as each
 * character is typed, the string from the text widget is compared with
 * all the strings in the list widget. The first string in the list widget,
 * that has the string in the text widget as the substring, is set to be
 * the first string in the list widget.
 */

void scll(Widget tw, XtPointer unused, XmTextVerifyCallbackStruct *cbs)
{
    char str[25];
    char *exp, *text;
    XmString pl;
    int cnt, j = 0;
    extern char *re_comp();
    char *p, *q;
    struct list *cur;
    p = (char *)malloc(sizeof(char)*1040);
    if (cbs->text->ptr == NULL)

```

```

        return;

/* Get the string in text Widget. */
q = XmTextGetString(tw);
strcat(q, "\0");
strcpy(p, q);
strncat(p, cbs->text->ptr, cbs->text->length);
strcat(p, "\0");
cnt = 14;

/* Compare the string to the names in the list. If a match is
 * found, set it to be the first visible item in the list.
 */

cur = listhead;
while (cur != NULL) {
    if (!strncmp(cur->name, p, strlen(p))) {
        strcpy(str, cur->name);
        strcat(str, "\0");
        pl = XmStringCreateSimple(str);
        XmListSetItem(list_w, pl);
        XmStringFree(pl);
    }
    cur = cur->right;
}
XtFree(p);
XtFree(q);
}

/* When an item in the list widget is double-clicked, the following function
 * is called. If the item is not a logical link and is found in the list
 * of file names, the description is shown in a dialog.
 */

void select_list(Widget w, XtPointer client_data, XmListCallbackStruct *cbs)
{
    int    count, pos = 0;
    struct list *p;
    pos = cbs->item_position;
    if (pos != 0) {
        p = listhead;
        for (count = 0; count < pos; count++)
            p = p->right;
        printf("pos = %s", p->name);
        if (!p->type) {
            /* Not a logical link, so display its description */
            show_desc(w, client_data, cbs);
        }
    }
    XtUnmanageChild(top);
}

/* This function is called when the user pushes the thesaurus button.
 * The word in the text widget is searched in the database and if a match
 * is found, its equivalent words are displayed in a dialog.
 */

void try_thesaurus(Widget tw, XtPointer unused, XmTextVerifyCallbackStruct *cbs)
{
    char    *q;
    q = XmTextGetString(pl);
    if (thesaurus(q)) {
        /* Word is found in the database, display the equivalent words*/
        alternate_words(toplevel, q);
    } else
        errordialog("word not found in thesaurus....");
}

/* This function is called when the enter key is pressed in the text widget
 * This function is used to display the file names of the files connected to

```

```

* the logical link typed in the text widget. If the word is not a logical
* and also not a valid file name, then an error message is issued.
*/

void text_select(Widget tw, XtPointer unused, XmTextVerifyCallbackStruct *cbs)
{
    char    *q;
    XmString r;
    struct list *p;
    char    args[100];
    int     count, pos = 0;

    q = XmTextGetString(tw);
    r = XmStringCreateSimple(q);
    pos = XmListItemPos(list_w, r);

    if (pos != 0) {
        p = listhead;
        for (count = 0; count < pos; count++)
            p = p->right;
        printf("pos = %s", p->name);

        if (p->type) {
            /* Post alternate list. */
            alternate_list(toplevel, p->name);
        }
    } else {
        errordialog("FILE NOT FOUND, TRY THESAURUS OPTION...");
    }
}

/* This function is called to load the file names and the logical links into
* the list widget.
*/

void list_names(Widget list_w)
{
    struct list *p, *q, *next;
    int     flag = 0;
    char    temp[80];
    XmString listname;
    int     i;

    /* Load the list of logical links from the file. */

    load_list();
    p = listhead;
    while (p->right != NULL)
        p = p->right;

    /* Add the list of file names to the above list. */

    add_list(p, head);
    i = 1;

    /* Sort the list using insertion sort. */

    p = listhead->right;
    while (p != NULL) {
        next = p->right;
        while (next != NULL) {
            if (strcmp(p->name, next->name) > 0) {
                memset(temp, '\0', 80);
                strcpy(temp, p->name);
                strcpy(p->name, next->name);
                strcpy(next->name, temp);
            }
            next = next->right;
        }
        p = p->right;
    }

    /* Add each member of the list to the list widget after
    * checking that there are no duplicates.
    */
}

```

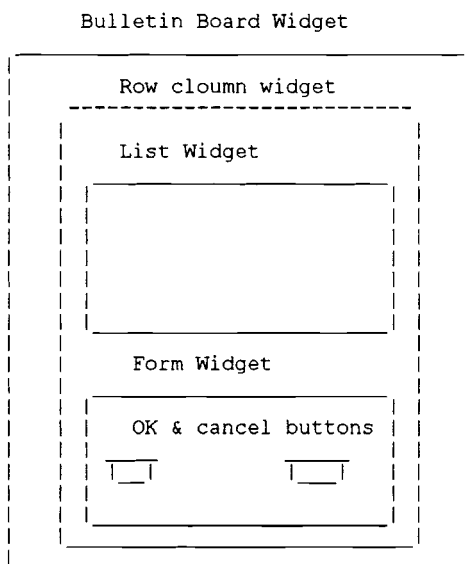
```

*/
p = listhead->right;
while (p != NULL) {
    q = listhead;
    for (; q == p; q = q->right) {
        if (strcmp(q->name, p->name) == 0)
            flag = 1;
    }
    if (!flag) {
        listname = XmStringCreateSimple(p->name);
        XmListAddItemUnselected(list_w, listname, i++);
    }
    p = p->right;
}
}

/* This function is called to display the alternate words found in the
 * thesaurus of the library in a dialog.
 * Double-clicking on a word in this list posts the word in the text
 * Widget of the list dialog after dismissing this dialog.
 */

/*
The structure of this widget is.

```



```

*/
void alternate_words(Widget w, char str[80])
{
    Arg args[10];
    int i = 0;
    char s[80];
    XmString one, two;
    Widget topl, ll, labell, list_w, row_col3, pb1, pb2;
    XtSetArg(args[i], XmNautoUnmanage, True);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNwidth, 400);
    i++;
    XtSetArg(args[i], XmNheight, 300);
    i++;

    topl = XmCreateBulletinBoardDialog(w, "LIST", args, 5);

    /* Position the dialog at 100,100. */

```

```

XtAddCallback(top1, XmNmapCallback, map, 100100);
XtAddCallback(top1, XmNhelpCallback, help, 5);

l1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, top1, XmNnumColumns,
    1, XmNorientation, XmVERTICAL, NULL);
sprintf(s, "ALTERNATE WORDS TO %s ", str);
one = XmStringCreateSimple(s);
label1 = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
    one, NULL);
XtFree(one);
list_w = XmCreateScrolledList(l1, "list_w", NULL, 0);
XtVaSetValues(list_w, XmNvisibleItemCount, 5, NULL);

/* Call the function set_text for any interaction in the list
 * Widget.
 */

XtAddCallback(list_w, XmNdefaultActionCallback, set_text, NULL);

rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, l1, NULL);
one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
    XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
    XmNleftOffset, 5, XmNheight, 25, NULL);
pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
    XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
    XmNrightOffset, 5, XmNheight, 25, NULL);
XtAddCallback(pb1, XmNactivateCallback, destroy, top1);
XtAddCallback(pb2, XmNactivateCallback, destroy, top1);
XtManageChild(rowcol3);
XtManageChild(list_w);
XtManageChild(label1);

/* This function is called to load the alternate words into
 * the list widget.
 */

load_words(list_w);
XtManageChild(l1);
XtManageChild(top1);
}

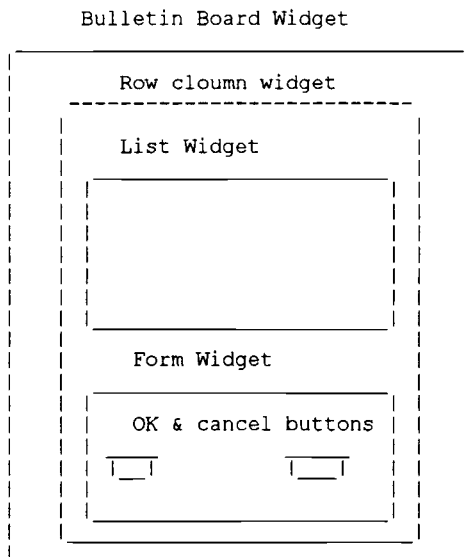
```

```

/* This function is called to display the files connected to the
 * logical link typed in the text widget. The file names are displayed
 * in a list. Double-clicking a file name in the list causes its
 * description to be displayed.
 */
/*

```

The structure of this widget is:



```

*/
void alternate_list(Widget w, char str[80])
{
    Arg args[10];
    int i = 0;
    char s[80];
    XmString one, two;
    Widget top1, l1, labell, list_w, pb1, pb2, rowcol3;
    XtSetArg(args[i], XmNautoUnmanage, False);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNwidth, 400);
    i++;
    XtSetArg(args[i], XmNheight, 300);
    i++;

    top1 = XmCreateBulletinBoardDialog(w, "LIST", args, 5);

    /* Position the dialog at 550,100. */

    XtAddCallback(top1, XmNmapCallback, map, 550100);
    XtAddCallback(top1, XmNhelpCallback, help, 4);

    l1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, top1, XmNnumColumns,
        1, XmNorientation, XmVERTICAL, NULL);
    sprintf(s, "FILES CONNECTED TO %s ", str);
    one = XmStringCreateSimple(s);
    labell = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
        one, NULL);
    XtFree(one);
    list_w = XmCreateScrolledList(l1, "list_w", NULL, 0);
    XtVaSetValues(list_w, XmNvisibleItemCount, 5, NULL);
    XtAddCallback(list_w, XmNdefaultActionCallback, show_desc, top1);

    rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, l1, NULL);
    one = XmStringCreateSimple("OK");
    two = XmStringCreateSimple("CANCEL");
    pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNheight, 25, NULL);

    pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
        XmNrightOffset, 5, XmNheight, 25, NULL);
    XtAddCallback(pb1, XmNactivateCallback, select_file, top1);
    XtAddCallback(pb2, XmNactivateCallback, destroy, top1);
    XtManageChild(rowcol3);

    XtManageChild(list_w);
    XtManageChild(labell);

    /* This function is called to load the file names into the list. */

    alternate_names(list_w, str);
    XtManageChild(l1);
    XtManageChild(top1);
}

/* This function is called to load the alternate words into the
 * thesaurus list widget.
 */
void load_words(Widget list)
{
    struct word list *cur;
    XmString listname;
    int i;

    /* Load each alternate word into the list. */

    cur = wordhead->right;
    i = 1;

```

```

    while (cur) {
        listname = XmStringCreateSimple(cur->word);
        XmListAddItemUnselected(list, listname, i++);
        cur = cur->right;
    }
}

/* This function is called to load the names of files connected
 * to the logical link.
 */

void alternate_names(Widget list_w, char str[80])
{
    struct list *p;
    char    s[100];
    XmString listname;
    int     i;
    load_list();
    p = listhead;
    while (p->right)
        p = p->right;
    add_list(p, head);
    i = -1;

    /* Load all file names connected to the logical link */

    p = listhead->right;
    while (p != NULL) {
        if (strcmp(str, p->name) == 0) {
            sprintf(s, "%s (%d)", p->filename, p->distance);
            listname = XmStringCreateSimple(s);
            XmListAddItemUnselected(list_w, listname, i++);
        }
        p = p->right;
    }
}

/* This function is called when the user interacts with any alternate word
 * in the thesaurus list. This word is posted in the text widget after
 * destroying the thesaurus dialog.
 */

void set_text(Widget tw, XtPointer unused, XmListCallbackStruct *cbs)
{
    char    *choice, *s;
    XmStringGetLtoR(cbs->item, charset, &choice);
    s = strtok(choice, " ");
    XmTextSetString(pl, s);
    XtFree(choice);
}

/* Double-clicking on any item in the list widget of link dialog calls
 * this function. The item is passed to description function to display
 * its description.
 */

void show_desc(Widget tw, XtPointer unused, XmListCallbackStruct *cbs)
{
    char    *choice, str[80];

    XmStringGetLtoR(cbs->item, charset, &choice);
    strcpy(def, choice);
    XtFree(choice);

    if (unused)
        XtUnmanageChild(unused);

    /* Show Description of the file. */

    choice = strtok(def, " ");
    strcpy(str, choice);
    description(str);
}

/* When the OK button in the description dialog is pushed, the following
 * function is called. This function destroys the description dialog and
 * posts the name of the file in the text widget of the List dialog to be

```

```

* considered as a candidate.
*/

void select_file(Widget tw, XtPointer w, XmListCallbackStruct *cbs)
{
    XtUnmanageChild(w);
    XmTextSetString(pl, def);
}

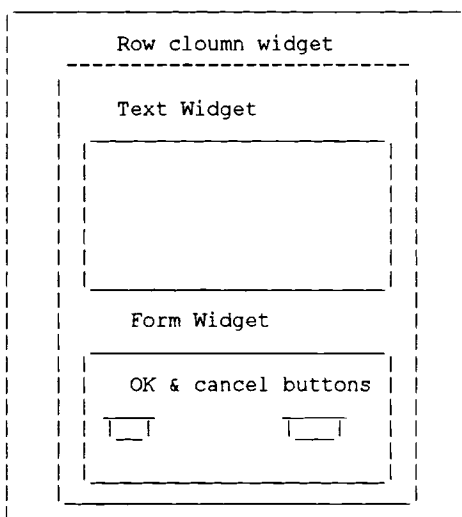
/* This function is called to display the description of a file.
* The file is located in the library and its description is retrieved.
* Its description is then formatted for display in a dialog.
*/

void description(char str[80])
{
    char    string[2000];
    struct list *cur;
    cur = listhead;
    while (cur && strcmp(cur->name, str) != 0)
        cur = cur->right;
    if (cur) {
        if (!cur->type) {
            point = NULL;
            traverse(head, str);
            if (point) {
                sprintf(string, "FUNCTION :%s \n METHOD : %s \n
                    IMPLEMENTATION:%s\n AUTHOR : %s\n EMAIL: %s \n",
                    point->function, point->method, point->implementation,
                    point->author, point->email);
                show_description(string);
            }
            else {
                errordialog("FILE NOT FOUND IN LIBRARY .....");
                return;
            }
        }
        else {
            errordialog("IT IS A LOGICAL LINK PRESS ENTER AND THEN TRY
                DESCRIPTION OPTION FOR CONNECTED FILES.... ");
            return;
        }
    }
    else {
        errordialog("ERROR .. not found in list.....");
        return;
    }
}

/* This function is called to display the description of a file.*/
/*
The structure of this widget is:

```

Bulletin Board Widget




```

*/

void show_description(char str[2000])
{
    Widget dialog, text_w1, rowcoll, form, pb1;
    XmString one, two;
    Arg args[10];
    int i;
    i = 0;
    XtSetArg(args[i], XmNautoUnmanage, False);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNwidth, 400);
    i++;
    XtSetArg(args[i], XmNheight, 300);
    i++;

    dialog = XmCreateBulletinBoardDialog(toplevel, "MESSAGE", args, 6);
    XtAddCallback(dialog, XmNmapCallback, map, 275350);

    rowcoll = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, dialog,
                               XmNnumColumns, 1, XmNorientation, XmVERTICAL, NULL);

    XtSetArg(args[0], XmNrows, 10);
    XtSetArg(args[1], XmNcolumns, 50);
    XtSetArg(args[2], XmNeditable, False);
    XtSetArg(args[3], XmNeditMode, XmMULTI_LINE_EDIT);
    XtSetArg(args[4], XmNwordWrap, True);
    XtSetArg(args[5], XmNvalue, str);
    text_w1 = XmCreateScrolledText(rowcoll, "text", args, 6);
    XtManageChild(text_w1);

    form = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcoll, XmNfractionBase,
                            7, NULL);

    one = XmStringCreateSimple("OK");
    two = XmStringCreateSimple("CANCEL");
    pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, form,
                                   XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
                                   XmNleftOffset, 5, XmNheight, 25, NULL);

    pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, form,
                                   XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
                                   XmNrightOffset, 5, XmNheight, 25, NULL);
    XtAddCallback(pb1, XmNactivateCallback, select_file, dialog);
    XtAddCallback(pb2, XmNactivateCallback, destroy, dialog);

    XtFree(one);
    XtManageChild(pb1);
    XtManageChild(rowcoll);
    XtManageChild(form);
    XtManageChild(dialog);
}

/* This function is called when the OK button in the list dialog is pushed
 * As mentioned before, the list dialog can pop up in two situations (refer
 * to list dialog comments at the beginning of this file). If the list was called
 * in the libin process, the file name is taken as the parent of the current file
 * being checked in, and if the user is the library administrator, then the actual
 * check-in process is initiated, else a note is made in the Experimental.dat
 * file.
 * If the list was displayed in the libout process, the selected file is checked
 * out of the library into the current directory of the user.
 */

void final_in(Widget tw, Widget wid_d, XmPushButtonCallbackStruct *cbs)
{
    XmString m;
    struct list *n;
    char args[100];
    int count, pos = 0;
    struct list *cur, *new;
    char *q, *p, s[80], r[80];
    int i = 0;

```

```

q = XmTextGetString(p1);
m = XmStringCreateSimple(q);
pos = XmListItemPos(list_w, m);
if (pos != 0) {
    n = listhead;
    for (count = 0; count < pos; count++)
        n = n->right;
    printf("pos = %s", n->name);
    if (n->type) {

        /* The word typed in the text widget is a logical link.
        * Display the file names to which it is connected.
        */

        alternate_list(toplevel, n->name);
    } else
    {
        if (!checkoutflag) {

            /* List called in LIBIN Process */

            strcpy(s, file_name);
            p = strtok(s, ".");
            q = strtok(NULL, "\\0\\n");
            if (p){
                sprintf(r, "rcs%s", p);
                strcat(r,q);
            }
            else
                sprintf(r, "rcs%s", file_name);

            strcpy(new_file->rcsno, r);
            strcpy(new_file->name, file_name);
            strcpy(src_d, dir);
            if (sys_adm) {

                /* The user is the library administrator
                * Proceed to actually check in
                * the file.
                */

                new_file->status = 1;
                add_node(new_file, q);
            } else {

                /* Include in Experimental.dat as the
                * user is not the Library_administrator
                */

                sprintf(s, "cp %s%s %s%s", src_d,
                    file_name, lib_d, file_name);
                system(s);
                memset(s, '\\0', 80);
                sprintf(s, "chmod 777 %s%s", lib_d,
                    file_name);
                system(s);
                write_exp(new_file, q);
                new_file->status = 0;
                add_exp(new_file, q);
            }
        } else {

            /* List displayed in Libout mode.
            * Checkout the file from the library.
            */

            checkout(q);
        }
        XtUnmanageChild(wid_d);
    }
} else
    errordialog("Cannot find file in list ");
}

```

```

/*
* * * * *
*   Filename : Libout.c
* * * * *
*   Programmed by : N.Sunil Chakravarthy.
* * * * *
*   Last updated on : 08.17.94
* * * * *
*/

```

This file contains the code to pop up the dialog asking the user to choose from the "File name selected from list" and "File name selected by pattern search" options.

```

*/
#include <Xm/DialogS.h>
#include <Xm/TextF.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/Form.h>
#include <Xm/SelectioB.h>
#include <Xm/PushB.h>
#include <Xm/LabelG.h>

void by_name(), pattern_search(), check_out(), search_desc(), select_name();

extern Widget rowcol;

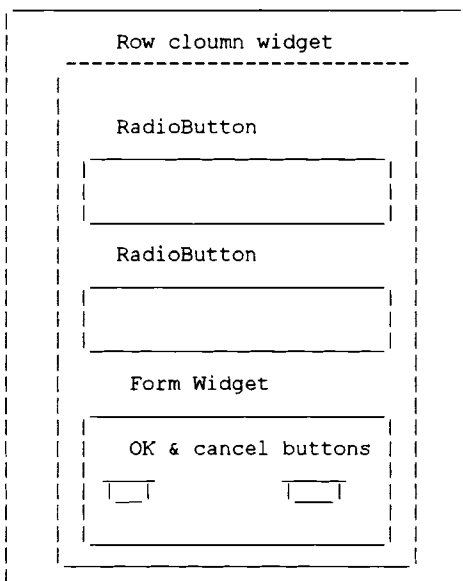
extern void destroy();
extern void Destroy();
extern void help();
extern void list_display();
extern pattern();
extern map();
extern toggle();
extern int mode;
extern Widget toplevel;

int checkoutflag;
Widget outdialog;

/* This function is called after the Libout button is pushed by the user.
 * This function allows the user to choose from "Filename selected
 * from list" and "Filename selected by pattern matching " options.
 */
/*

```

The structure of this dialog box is as follows.
Bulletin Board Widget



```

*/

void lib_out(Widget w)
{
    Widget pb1, cin_R, cin_N, cin_S, cin_T, cin_M, rowcol1, pb2, rowcol3,
        rad;
    XmString one, two, three, four, five;
    Arg args[10];
    int i;

    i = 0;
    checkoutflag = 0;
    XtSetArg(args[i], XmNautoUnmanage, True);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNwidth, 100);
    i++;
    XtSetArg(args[i], XmNheight, 100);
    i++;
    XtSetArg(args[i], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);
    outdialog = XmCreateBulletinBoardDialog(toplevel, "LIBOUT", args, 6);

    /* Position the dialog at 150,125. */

    XtAddCallback(outdialog, XmNmapCallback, map, 150125);
    XtAddCallback(outdialog, XmNhelpCallback, help, 6);

    rowcol1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, outdialog,
        XmNnumColumns, 1, XmNpacking, XmPACK_COLUMN, XmNorientation,
        XmVERTICAL, NULL);
    mode = 0;

    /* Create the radiobuttons. */

    one = XmStringCreateSimple("FILE SELECTED BY NAME");
    two = XmStringCreateSimple("FILE SELECTED BY PATTERN MATCHING");
    rad = XmVaCreateSimpleRadioBox(rowcol1, "radio_box", 0, toggle, XmVaRADIOBUTTON,
        one, 0, NULL, NULL, XmVaRADIOBUTTON, two, 0, NULL, NULL,
        NULL);

    rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcol1,
        NULL);

    one = XmStringCreateSimple("OK");
    two = XmStringCreateSimple("CANCEL");
    pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNbottomAttachment, XmATTACH_FORM, XmNleftOffset, 5,
        XmNheight, 25, NULL);

    pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
        XmNbottomAttachment, XmATTACH_FORM, XmNrightOffset,
        5, XmNheight, 25, NULL);

    XtAddCallback(pb1, XmNactivateCallback, check_out, outdialog);
    XtAddCallback(pb2, XmNactivateCallback, destroy, outdialog);

    XmStringFree(one);
    XmStringFree(two);
    XtManageChild(rowcol1);
    XtManageChild(rowcol3);
    XtManageChild(rad);
    XtManageChild(outdialog);

    return;
}

/* File name selected by pattern search was pushed, record it. */
void pattern_search(Widget w_id, XtPointer client_data, XmPushButtonCallbackStruct *cbs)
{

```

```

    mode = 1;
}

/* File name selected from list was pushed, record it. */
void by_name(Widget w_id, XtPointer client_data, XmPushButtonCallbackStruct *cbs)
{
    mode = 0;
}

void check_out(Widget w_id, XtPointer client_data, XmPushButtonCallbackStruct *cbs)
{
    if (mode) {
        /* Filename selected by pattern search was pushed
        * get the pattern.
        */
        XtUnmanageChild(outdialog);
        select_name();
    } else {
        /* File name selected from list was pushed,
        * Display the list.
        */
        XtUnmanageChild(outdialog);
        checkoutflag = 1;
        list_display(rowcol);
    }
}

/* This function is used to display a dialog to obtain
* the pattern to be used from the user.
*/

```

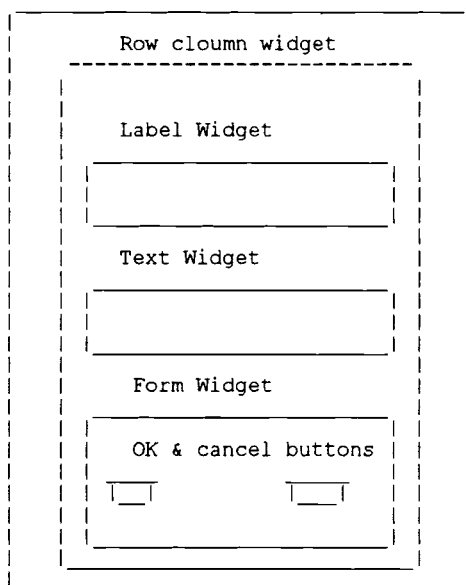
```

/*

```

The structure of this dialog box is as follows.

Bulletin Board Widget



```

*/

```

```

Widget pl;

```

```

void select_name()
{
    Widget      top, pb1, pb2, rowcol3, labell, l1;
    Arg  args[10];
    int   i = 0;
    XmString one, two;
    XtSetArg(args[i], XmNautoUnmanage, True);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNwidth, 400);
    i++;
    XtSetArg(args[i], XmNheight, 300);
    i++;
    XtSetArg(args[i], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);

    top = XmCreateBulletinBoardDialog(rowcol, "LIST", args, 6);
    XtAddCallback(top, XmNmapCallback, map, 200200);
    XtAddCallback(top, XmNhelpCallback, help, 7);

    l1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, top, XmNnumColumns,
        1, XmNorientation, XmVERTICAL, NULL);

    one = XmStringCreateSimple("Give Pattern :");
    labell = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
        one, NULL);

    p1 = XtVaCreateWidget("text", xmTextWidgetClass, l1, NULL);
    XtAddCallback(p1, XmNactivateCallback, search_desc, p1);

    rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, l1, NULL);
    one = XmStringCreateSimple("OK");
    two = XmStringCreateSimple("CANCEL");
    pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNheight, 25, NULL);

    pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
        XmNrightOffset, 5, XmNheight, 25, NULL);
    XtAddCallback(pb1, XmNactivateCallback, search_desc, p1);
    XtAddCallback(pb2, XmNactivateCallback, destroy, top);
    XtManageChild(rowcol3);
    XtManageChild(labell);
    XtManageChild(p1);
    XtManageChild(l1);
    XtManageChild(top);
}

/* Once the OK button in the above dialog is pushed, the following function is called.
 * After recording the text in the text widget as the required pattern, the function
 * pattern is called to search the pattern in the description of the files.
 */

void search_desc(Widget tw, XtPointer w, XmTextVerifyCallbackStruct *cbs)
{
    char *text;
    text = XmTextGetString(p1);

    XtUnmanageChild(XtParent(XtParent(w)));

    /* Function to search a pattern. */

    pattern(text);
}

```

```

/*
* * * * *
*   Filename : Pattern.c
*
*   Programmed by : N. Sunil Chakravarthy
*
*   Last updated on : 08.17.94
* * * * *
*/

```

This file contains the code to find the files whose description has the pattern given by the user in the Libout process.

```

*/
#include <stdio.h>
# include <ctype.h>
# include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <math.h>
#include <Xm/List.h>
# include <Xm/RowColumn.h>
# include <Xm/Text.h>
#include <Xm/DialogS.h>
#include <Xm/LabelG.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>

extern struct file_node {
    char   name[20];
    char   rcsfile[80];
    char   rcsno[2000];
    char   author[20];
    char   email[20];
    char   function[100];
    char   method[100];
    char   implementation[1000];
    float  saved;
    int    outno;
    int    status;
    int    filesize;
    struct file_node *left;
    struct file_node *right;
    struct file_node *parent;
};
extern struct file_node *head, *point;
extern XmStringCharSet charset;
extern Widget rowcol;
extern Widget pl;
void match_desc();
extern errOrdialog();
extern destroy();
extern help();
extern map();
extern traverse();
extern checkout();

struct match_list {
    char   name[20];
    struct match_list *right;
};
struct match_list *matchhead, *prev;
int   compare();
void match_display();
void match_names();
void match_traverse();
void match_show();
void match_file();
void match_desc();
void match_check();
void pattern_checkout();

extern char def[1000];

/* This function is called after the pattern is given by the user. */

```

```

int    pattern(char str[100])
{
    struct file_node *cur;
    cur = head;
    matchhead = (struct match_list *)malloc(sizeof(struct match_list ));
    if (!matchhead) {
        errorialog("Memory allocation problem in pattern matching ...");
        return;
    }
    matchhead->right = NULL;
    prev = matchhead;

    /* Check the tree recursively to see if any file description has the
     * required pattern. Make a linked list of all matching files.
     */

    match_traverse(head->left, str);
    match_display(rowcol);
}

/* This function is used to check the tree recursively to see if any file's
 * description has the required pattern. A linked list is made of the
 * matching files.
 */

void match_traverse(struct file_node *cur, char str[100])
{
    int    match = 0;
    struct match_list *point;
    if (cur) {

        /* Compare all portions of the description. */

        match = compare(cur->function, str);
        if (!match)
            match = compare(cur->method, str);
        if (!match)
            match = compare(cur->implementation, str);
        if (!match)
            match = compare(cur->author, str);
        if (!match)
            match = compare(cur->email, str);
        if (match) {

            /* File matched, add to linked list */

            point = (struct match_list *)malloc(sizeof(struct match_list ));
            if (!point) {
                errorialog("Memory allocation problem in pattern matching
                ...");
                return;
            }
            strcpy(point->name, cur->name);
            prev->right = point;
            point->right = NULL;
            prev = point;
        }
        match_traverse(cur->left, str);
        match_traverse(cur->right, str);
    }
}

/* This function is used to see if a given pattern is found in a given string. */

int    compare(char str1[500], char str2[100])
{
    int    i = 0, ret_val = 0, j = 0;
    char    token[500];
    while (str1[i] != '\0' && !ret_val) {
        memset(token, '\0', 500);
        while (str1[i] != ' ' && str1[i] != '\0') {
            token[j] = str1[i];
            j++;
            i++;
        }

        /* If pattern is found, set ret_val to 1. */
    }
}

```



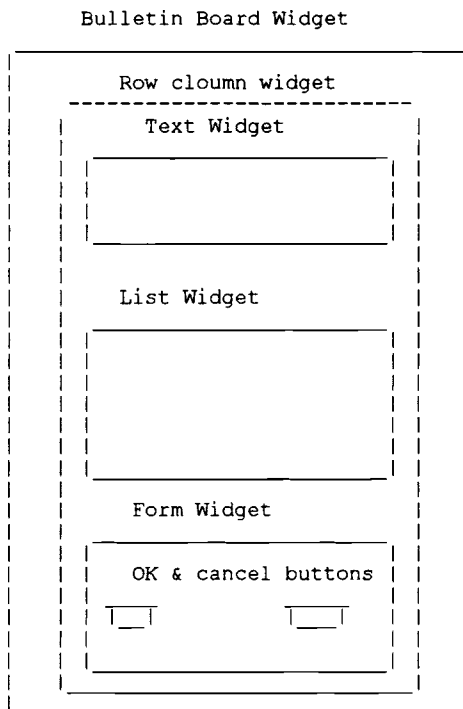
```

        if (strcmp(str2, token) == 0)
            ret_val = 1;
        while (str1[i] == ' ' && str1[i] != '\0')
            i++;
        j = 0;
    }
    return(ret_val);
}

/* This function is called to display the list of files with
 * description having the required pattern.
 */

/*
    The structure of this widget is:

```



```

*/

void match_display(Widget w)
{
    Widget      top, pb1, pb2, pb3, rowcol3, list_w, sb, labell, ll;

    Arg  args[10];
    int  i = 0;
    XmString one, two;
    XtSetArg(args[i], XmNautoUnmanage, True);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNx, 375);
    i++;
    XtSetArg(args[i], XmNy, 100);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNwidth, 400);
    i++;
    XtSetArg(args[i], XmNheight, 300);
    i++;
    XtSetArg(args[i], XmNuserData, 0);

    top = XmCreateBulletinBoardDialog(w, "LIST", args, 7);
    XtAddCallback(top, XmNmapCallback, map, 350125);
}

```

```

XtAddCallback(top, XmNhelpCallback, help, 8);

l1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, top, XmNnumColumns,
    1, XmNorientation, XmVERTICAL, NULL);

one = XmStringCreateSimple("SELECT FILE:");
labell = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
    one, NULL);

p1 = XtVaCreateWidget("text", xmTextWidgetClass, l1, NULL);

list_w = XmCreateScrolledList(l1, "list_w", NULL, 0);
XtVaSetValues(list_w,
    XmNvisibleItemCount, 5, NULL);
XtAddCallback(list_w, XmNdefaultActionCallback, match_desc, NULL);

rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, l1, NULL);
one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
    XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
    XmNleftOffset, 5, XmNheight, 25, NULL);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
    XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
    XmNrightOffset, 5, XmNheight, 25, NULL);
XtAddCallback(pb1, XmNactivateCallback, pattern_checkout, p1);
XtAddCallback(pb2, XmNactivateCallback, destroy, top);
XtManageChild(rowcol3);
XtManageChild(list_w);
XtManageChild(labell);
XtManageChild(p1);

/* Call this function to load the files into the list widget. */

match_names(list_w);
XtManageChild(l1);
XtManageChild(top);
}

/* This function is used to load the list of files, whose description has the
 * required pattern, into the list widget.
 */

void match_names(Widget w)
{
    struct match_list *p, *q;
    int flag = 0;
    XmString listname;
    int i;
    i = 1;

    /* Traverse the linked list of file names already chosen,
     * and add one by one to the list
     */

    p = matchhead->right;
    while (p) {
        q = matchhead;
        for (; q == p; q = q->right) {
            if (strcmp(q->name, p->name) == 0)
                flag = 1;
        }
        if (!flag) {
            listname = XmStringCreateSimple(p->name);
            XmListAddItemUnselected(w, listname, i++);
            XtFree(listname);
        }
        p = p->right;
    }
}

/* This function is called when the user double-clicks
 * a file name in the list. This function calls match_check with
 * the name of the file.
 */

```

```

void match_desc(Widget tw, XtPointer unused, XmListCallbackStruct *cbs)
{
    char    *choice, str[80];

    XmStringGetLtoR(cbs->item, charset, &choice);
    strcpy(def, choice);

    XtFree(choice);
    /* Show Description of the file */
    strcpy(str, def);
    match_check(str);
}

```

```

/* This function checks to see if the file exists in the library.
 * If so, it calls the match-show function to display its
 * description.
 */

```

```

void match_check(char str[80])
{
    char    string[2000];
    point = NULL;
    traverse(head->left, str);
    if (point) {
        sprintf(string, "FUNCTION :%s \n METHOD : %s \n IMPLEMENTATION :%s\n AUTHOR
                        : %s\n
                        EMAIL: %s \n",
                point->function, point->method, point->implementation, point->author,
                point->email);
        match_show(string);
    } else {
        errorDialog("FILE NOT FOUND IN LIBRARY .....");
        return;
    }
}

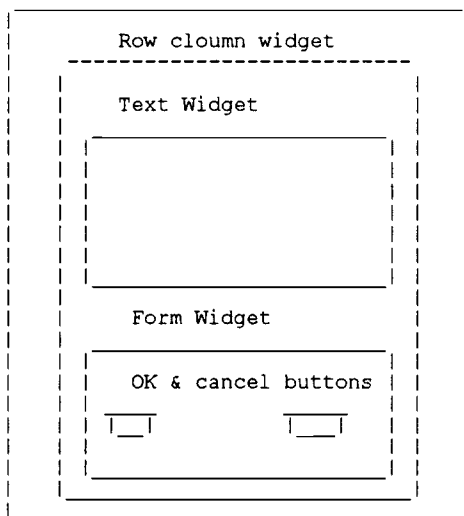
```

```

/* This function is called to display the description of a file.*/
/*
The structure of this widget is:

```

Bulletin Board Widget



```

*/

```

```

void match_show(char str[2000])
{
    Widget dialog, text_wl, rowcoll, form, pb1, pb2;
    XmString one, two;
    Arg args[10];
    int    i;
    i = 0;
    XtSetArg(args[i], XmNautoUnmanage, True);

```

```

i++;
XtSetArg(args[i], XmNdefaultPosition, False);
i++;
XtSetArg(args[i], XmNx, 275);
i++;
XtSetArg(args[i], XmNy, 350);
i++;
XtSetArg(args[i], XmNallowShellResize, True);
i++;
XtSetArg(args[i], XmNwidth, 400);
i++;
XtSetArg(args[i], XmNheight, 300);
i++;
dialog = XmCreateDialogShell(rowcol, "MESSAGE", args, 7);

rowcoll = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, dialog,
                          XmNnumColumns, 1, XmNorientation, XmVERTICAL, NULL);

XtSetArg(args[0], XmNrows, 10);
XtSetArg(args[1], XmNcolumns, 50);
XtSetArg(args[2], XmNeditable, False);
XtSetArg(args[3], XmNeditMode, XmMULTI_LINE_EDIT);
XtSetArg(args[4], XmNwordWrap, True);
XtSetArg(args[5], XmNvalue, str);
XtSetArg(args[6], XmNscrollHorizontal, False);
text_w1 = XmCreateScrolledText(rowcoll, "text", args, 7);
XtManageChild(text_w1);

form = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcoll, XmNfractionBase,
                       7, NULL);

one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, form,
                              XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
                              XmNleftOffset, 5, XmNheight, 25, NULL);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, form,
                              XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
                              XmNrightOffset, 5, XmNheight, 25, NULL);
XtAddCallback(pb1, XmNactivateCallback, match_file, dialog);
XtAddCallback(pb2, XmNactivateCallback, destroy, dialog);

XtFree(one);
XtManageChild(pb1);
XtManageChild(rowcoll);
XtManageChild(form);
}
/* When the ok button in the description dialog is pushed, the following
 * function is called. This function destroys the description dialog and
 * posts the name of the file in the text widget of the List dialog to be
 * considered as a candidate file to be checked out.
 */

void match_file(Widget tw, XtPointer w, XmListCallbackStruct *cbs)
{
    XtDestroyWidget(w);
    XmTextSetString(p1, def);
}

/* This function is called when the user pushes the OK button in the List
 * dialog. The text in the text widget is taken as the name of the file
 * to be checked out and the corresponding check-out procedures are
 * initiated.
 */

void pattern_checkout(Widget tw, XtPointer w, XmListCallbackStruct *cbs)
{
    char *p;
    p = XmTextGetString(w);
    checkout(p);
}

```

```

/*
* * * * *
*   Filename : Link.c
*
*   Programmed by : N.Sunil Chakravarthy.
*
*   Last updated on : 08.17.94
* * * * *
*/

This file contains the code to accept the name of the link, the file to
which it must be connected, and the distance.
*/

#include <Xm/DialogS.h>
#include <Xm/TextF.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/Form.h>
#include <Xm/SelectioB.h>
#include <Xm/PushB.h>
#include <Xm/LabelG.h>
#include <stdio.h>

void start_link();
void map();
extern destroy();
extern writelist();
extern free_list();
extern DestRoy();
extern help();
extern traverse();
extern errordialog();
extern lock();
extern unlock();
extern modal_error_dialog();

extern struct list {
    char    name[20];
    char    filename[20];
    int     type;
    int     distance;
    struct list *left;
    struct list *right;
};

extern point;
extern head;
extern Widget toplevel;
extern int    sys_adm;
extern char   lib_d[100];
extern char   *passwd;
extern struct list *listhead;
Widget link2, link3;

/* This function is called when the user pushes the link drawn button.
 * This function pops up a dialog with three text widgets to receive the name
 * of the link, the name of the file to which it is connected, and the
 * distance.
 */

void link(Widget w)
{
    Widget top, pb1, pb2, rowcol3, label1, l1, p1, label2, label3;
    Arg    args[10];
    int    i = 0, x = 0, y = 0;
    XmString one, two;
    XtSetArg(args[i], XmNautoUnmanage, True);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);
    top = XmCreateBulletinBoardDialog(toplevel, "LIST", args, 4);

```

```

/* Position the dialog at 100,100. */

XtAddCallback(top, XmNmapCallback, map, 100100);
XtAddCallback(top, XmNhelpCallback, help, 12);

l1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, top, XmNnumColumns,
    1, XmNorientation, XmVERTICAL, NULL);

one = XmStringCreateSimple("Give the name of the link:");
label1 = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
    one, NULL);

XtFree(one);
p1 = XtVaCreateWidget("text", xmTextWidgetClass, l1, NULL);
XtAddCallback(p1, XmNactivateCallback, XmProcessTraversal,
    XmTRAVERSE_NEXT_TAB_GROUP);

one = XmStringCreateSimple("Give the name of the File to be attached to:");
label2 = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
    one, NULL);

XtFree(one);
link2 = XtVaCreateWidget("text", xmTextWidgetClass, l1, NULL);
XtAddCallback(link2, XmNactivateCallback, XmProcessTraversal,
    XmTRAVERSE_NEXT_TAB_GROUP);

one = XmStringCreateSimple("Give the distance(0-10):");
label3 = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
    one, NULL);

XtFree(one);
link3 = XtVaCreateWidget("text", xmTextWidgetClass, l1, NULL);
XtAddCallback(link3, XmNactivateCallback, XmProcessTraversal,
    XmTRAVERSE_NEXT_TAB_GROUP);

rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, l1, NULL);
one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
    XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
    XmNleftOffset, 5, XmNheight, 25, NULL);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
    XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
    XmNrightOffset, 5, XmNheight, 25, NULL);
XtAddCallback(pb1, XmNactivateCallback, start_link, p1);
XtAddCallback(pb2, XmNactivateCallback, destroy, top);
XtManageChild(rowcol3);
XtManageChild(label1);
XtManageChild(p1);
XtManageChild(label2);
XtManageChild(link2);
XtManageChild(label3);
XtManageChild(link3);
XtManageChild(l1);
XtManageChild(top);
}

/* This function is used to position the dialog at the required coordinates.
 * the coordinates are sent as xy as one number. it is assumed the coordinates
 * are thriple digitd. To seperate x, divide (xy/1000) and take the integer
 * part. To seperate y, use xy- x*1000.
 */

void map(Widget tw, int w, XmAnyCallbackStruct *cbs)
{
    int    xpos = 0, ypos = 0, x1 = 0, y1 = 0;
    x1 = w / 1000;
    y1 = w - x1 * 1000;

    /* Get the coordinates of the toplevel shell. */

    XtVaGetValues(toplevel, XmNx, &xpos, XmNy, &ypos, NULL);
    XtVaSetValues(tw, XmNx, xpos + x1, XmNy, ypos + y1, NULL);
}

```

```

}

/* This function is called when the OK button is pushed in the link dialog.
 * This function first verifies if the distance is between 0 and 10.
 * Then the file name is checked to see if any file with the given name
 * exists in the library.
 * Then the link is checked to see if any link with the same name exists to the same file.
 * Finally, the link is appended to the linked list.
 */

void start_link(Widget tw, XtPointer w, XmTextVerifyCallbackStruct *cbs)
{
    struct list *cur, *new;
    char *text, *text1, *text2, str[100];
    FILE *fp;
    text = XmTextGetString(w);
    text1 = XmTextGetString(link2);
    text2 = XmTextGetString(link3);

    /* Verify if distance is between 0 to 10. */

    if (atoi(text2) > 10 || atoi(text2) < 0)
        modal_error_dialog(tw, "Distance should be between 0 and 10.");
    else {
        Destroy(XtParent(XtParent(w)));

        /* Check to see if the file exists in the library. */

        point = NULL;
        traverse(head, text1);
        if (!point) {
            errorialog("FILE NOT IN LIBRARY .....");
            return;
        }
        load_list();
        cur = listhead;

        /* Function to set links. */

        while (cur) {
            if (strcmp(cur->name, text) == 0) {
                if (strcmp(cur->filename, text1) == 0) {

                    /* Duplicate link give warning. */

                    sprintf(str, "THIS LINK ALREADY EXISTS WITH DISTANCE
                                %d", cur->distance);
                    errorialog(str);
                }
                return;
            }
            cur = cur->right;
        }
        cur = listhead;
        while (cur->right != NULL)
            cur = cur->right;

        /* Create a new link. */

        new = (struct list *)malloc(sizeof(struct list));
        if (!new) {
            errorialog("NOT ENOUGH SPACE TO MALLOC .....");
            return;
        }
        strcpy(new->name, text);
        strcpy(new->filename, text1);
        new->distance = atoi(text2);
        new->type = 1;
        cur->right = new;
        new->right = NULL;
        if (!sys_admin) {

            /* If not library administrator, make a note in the update.dat
             file. */
            /* Note that the change is already incorporated into the system.
             * Information is provided only for the knowledge of the library
             * administrator.
             */

```

```

sprintf(str, "%supdate.dat", lib_d);
fp = fopen(str, "r+");
if (!fp) {
    errordialog("UNABLE TO OPEN update file ....");
} else {
    if (lock(fp)) {
        fseek(fp, 0, 2);
        sprintf(str, "NEW SYMBOLIC LINK FROM %s to %s with
            distance %d added by %s\n",
            new->name, new->filename, new->distance,
            passwd);
        fputs(str, fp);
        unlock(fp);
        fclose(fp);
    } else
        errordialog("UNABLE TO LOCK update file ..Operation
            unsuccessful");
    }
}

writelist(listhead->right);
free_list(listhead);
}
}
}

```

```
/*
```

```

* * * * *
*   Filename : Display.c
*
*   Programmed by : N. Sunil Chakravarthy
*
*   Last updated on : 08.17.94
* * * * *

```

This file contains the code to pop up the dialog to receive the file name from which to start the display.

```
*/
```

```

#include <Xm/DialogS.h>
#include <Xm/TextF.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/Form.h>
#include <Xm/SelectioB.h>
#include <Xm/PushB.h>
#include <Xm/LabelG.h>

extern struct file_node {
    char   name[20];
    char   rcsfile[80];
    char   rcsno[2000];
    char   author[20];
    char   email[20];
    char   function[100];
    char   method[100];
    char   implementation[1000];
    float  saved;
    int    outno;
    int    status;
    int    filesize;
    struct file_node *left;
    struct file_node *right;
    struct file_node *parent;
};

extern struct file_node *point, *head;
struct file_node *child, *cur;

void start_display();
extern traverse();
extern Widget show();
extern tree1();

```



```

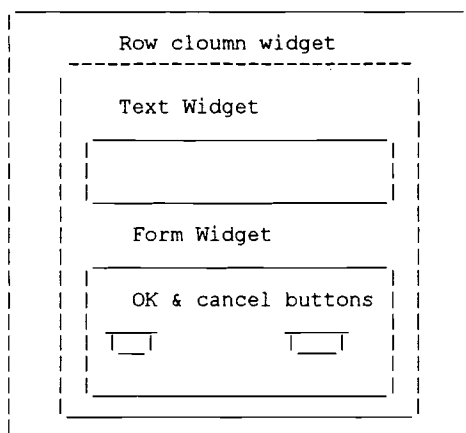
extern Widget rowcol;
extern destroy();
extern Destroy();
extern help();
extern map();

/* This function is called when the display drawn button is pushed.
 * This function pops up a dialog box with a text widget requesting the
 * user to enter the filename from which to start the display.
 */

/*
  The structure of this widget is as follows.

```

Bulletin Board Widget



```

*/

void display(Widget w)
{
    Widget      top, pb1, pb2, rowcol3, labell, l1, pl;
    Arg  args[10];
    int   i = 0;
    XmString one, two;
    XtSetArg(args[i], XmNautoUnmanage, False);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNwidth, 400);
    i++;
    XtSetArg(args[i], XmNheight, 300);
    i++;
    XtSetArg(args[i], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);

    top = XmCreateBulletinBoardDialog(w, "LIST", args, 6);

    /* Position the dialog at 150,150. */

    XtAddCallback(top, XmNmapCallback, map, 150150);
    XtAddCallback(top, XmNhelpCallback, help, 9);

    l1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, top, XmNnumColumns,
        1, XmNorientation, XmVERTICAL, NULL);

    one = XmStringCreateSimple("Give the name Of the File from which to
        display(head):");
    labell = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
        one, NULL);

    XtFree(one);
    pl = XtVaCreateWidget("text", xmTextWidgetClass, l1, NULL);
    XtAddCallback(pl, XmNactivateCallback, start_display, pl);

```

```

rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, ll, NULL);
one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
                               XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
                               XmNleftOffset, 5, XmNheight, 25, NULL);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
                               XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
                               XmNrightOffset, 5, XmNheight, 25, NULL);
XtAddCallback(pb1, XmNactivateCallback, start_display, p1);
XtAddCallback(pb2, XmNactivateCallback, destroy, top);
XtManageChild(rowcol3);
XtManageChild(label1);
XtManageChild(p1);
XtManageChild(ll);
XtManageChild(top);
}

/* This function is called once the OK button is pushed in the above
 * dialog. This function verifies if the file name entered in the above
 * dialog is present in the library. If the file is not present,
 * the display is started from the head node. The function to display
 * the tree is then called.
 */

void start_display(Widget tw, XtPointer w, XmTextVerifyCallbackStruct *cbs)
{
    char *text;
    Widget new;
    text = XmTextGetString(w);

    /* Function to display. */

    XtUnmanageChild(XtParent(XtParent(w)));
    new = show();
    point = NULL;
    traverse(head, text);
    if (point) {
        /* The filename entered is valid. Set child to the first child
         * of this node and set cur to point to this node to start the
         * display.
         */
        child = point->left;
        cur = point;
        treel(new);
    } else {
        /* The filename entered is invalid. Start the display from the
         * head node.
         */
        point = head;
        child = point->left;
        cur = point;
        treel(new);
    }
}

/*
 * * * * *
 *   Filename : Show.c
 * * * * *
 *   Programmed by : N. Sunil Chakravarthy
 * * * * *
 *   Last updated on : 08.17.94
 * * * * *
 */

```

This file has the code for the zoom_in and the zoom_out mode of the display.

```

*/

#include <Xm/List.h>
# include <Xm/RowColumn.h>
# include <Xm/Form.h>
# include <Xm/PushB.h>
# include <Xm/Text.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <stdio.h>
#include <stdlib.h>
# include <sys/wait.h>
# include <sys/types.h>
#include <unistd.h>

# define LEFT 0
# define RIGHT 1
# define REQLEFT 2
# define REQRIGHT 3
# define TOP 4
# define BOTTOM 5
# define z_in 0
# define z_out 1
# define diameter 20

extern struct file_node {
    char    name[20];
    char    rcsfile[80];
    char    rcsno[2000];
    char    author[20];
    char    email[20];
    char    function[100];
    char    method[100];
    char    implementation[1000];
    float   saved;
    int     outno;
    int     status;
    int     filesize;
    struct  file_node *left;
    struct  file_node *right;
    struct  file_node *parent;
};

struct display {
    struct  file_node *point;
    int     shape;
    int     depth;
    int     nodes;
    int     sibling_no;
};

struct point {
    int     x;
    int     y;
};

struct point  mypoint[6][6];
struct display mydisplay[8][6];
int           left_level = 0, right_level = 0, mode = 0, zoom_end = 0, maxdepth = 0,
             curdepth = 0, node_count = 0;

struct file_node *right_pointer, *gl_node;
extern char    lib_d[100];
extern struct file_node *child, *point, *head, *cur;
struct file_node *childold, *childnew, *top;

void draw_callback();
void draw_arrow();
void zoom_out();
void zoom_in();
void display_tree();
void initialize();
void redraw();
void map();
void depth();
void nodes();
void load_depth();
void m_depth();

```

```

void draw_legend();
void rdraw();
struct file_node *parent();
int sibling();
int test();
void reset();

```

```

Pixmap z_in_pixmap, z_out_pixmap;
GC gc;

```

```

extern help();
extern destroy();
extern Widget toplevel;

```

```

/* This function is called as soon as the file name from which the display is
 * to be started is given by the user. This function creates a dialog with
 * a drawing area widget and four pushbuttons. This function returns the
 * dialog widget.
 */

```

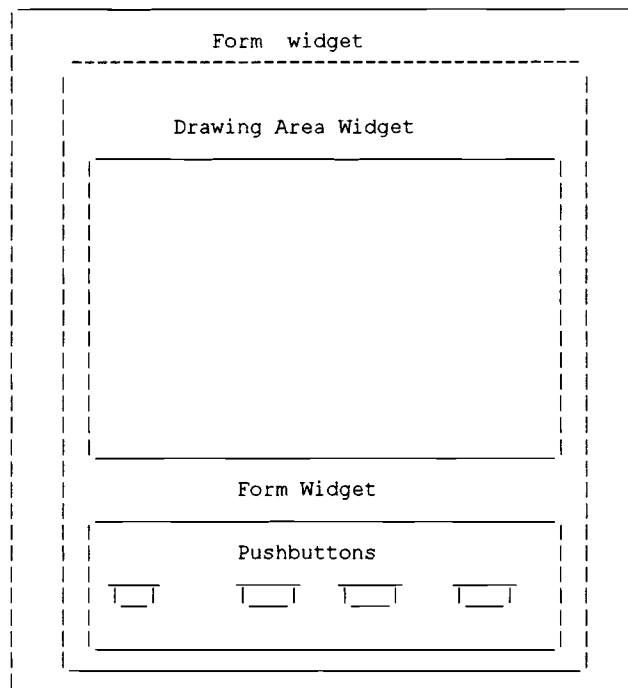
```

/*

```

The structure of this widget is as follows.

Bulletin Board Widget



```

*/

```

```

Widget show(w)
{

```

```

    Arg args[14];
    XmString one, two, three;
    Widget top, form1, ll, draw_w, pb1, pb2, pb3, pb4;
    int i;
    right_level = 0;
    left_level = 0;
    mode = 0;
    zoom_end = 0;
    XtSetArg(args[0], XmNautoUnmanage, False);
    XtSetArg(args[1], XmNdefaultPosition, False);
    XtSetArg(args[2], XmNallowShellResize, True);
    XtSetArg(args[3], XmNwidth, 800);
    XtSetArg(args[4], XmNheight, 700);

```

```

    top = XmCreateBulletinBoardDialog(toplevel, "LIST", args, 5);

```

```

/* Position the dialog at 125,15. */

XtAddCallback(top, XmNmapCallback, map, 125015);
XtAddCallback(top, XmNhelpCallback, help, 10);

form1 = XtVaCreateManagedWidget("form", xmFormWidgetClass, top, NULL);
l1 = XtVaCreateManagedWidget("rowcol", xmFormWidgetClass, form1,
    XmNbottomAttachment,
    XmATTACH_FORM, XmNleftAttachment, XmATTACH_FORM, XmNrightAttachment,
    XmATTACH_FORM, XmNfractionBase, 5, NULL);
draw_w = XtVaCreateManagedWidget("dbutton", xmDrawingAreaWidgetClass, form1,
    XmNtopAttachment, XmATTACH_FORM, XmNbottomAttachment, XmATTACH_WIDGET,
    XmNbottomWidget, l1, XmNwidth, 800, XmNheight, 600, XmNborderWidth, 2,
    XmNshadowThickness, 1, NULL);

z_in_pixmap = XCreatePixmap(XtDisplay(draw_w),
    RootWindowOfScreen(XtScreen(draw_w)),
    700, 600, DefaultDepthOfScreen(XtScreen(draw_w)));
z_out_pixmap = XCreatePixmap(XtDisplay(draw_w),
    RootWindowOfScreen(XtScreen(draw_w)),
    700, 600, DefaultDepthOfScreen(XtScreen(draw_w)));

XtAddCallback(draw_w, XmNinputCallback, draw_callback, NULL);
XtAddCallback(draw_w, XmNexposeCallback, redraw, NULL);

one = XmStringCreateSimple("ZOOM OUT");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, l1, XmNlabelString,
    one, XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1, NULL);
XtFree(one);

one = XmStringCreateSimple("ZOOM IN");
pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, l1, XmNlabelString,
    one, XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 2, NULL);
XtFree(one);

one = XmStringCreateSimple("LEGEND");
pb4 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, l1, XmNlabelString,
    one, XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 3, NULL);
XtFree(one);

one = XmStringCreateSimple("EXIT");
pb3 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, l1, XmNlabelString,
    one, XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 4, NULL);
XtFree(one);

/* Connect the functions to be called to the Pushbuttons. */

XtAddCallback(pb1, XmNactivateCallback, zoom_out, draw_w);
XtAddCallback(pb2, XmNactivateCallback, zoom_in, draw_w);
XtAddCallback(pb3, XmNactivateCallback, reset, top);
XtAddCallback(pb4, XmNactivateCallback, draw_legend, top);
XtManageChild(top);

/* Return the drawing area widget which will be used in other
 * functions to draw.
 */

return(draw_w);
}

/* This function handles the zoom in mode of the display.
 * The pointer to the current node is taken from the global variable cur.
 * If there is a current node, display it as a rectangle in the center.
 * Display its name and status. Similarly, show three children of this node
 * if present. If there are more than three children, show an arrow in that
 * direction at the edge of the box. Display a number at the edge of the
 * box near the arrow to indicate the number of children remaining to be
 * seen in that direction. Check if the current node has a parent. If so,
 * display the parent to the left of the current node.
 */

void tree1 (Widget wid)
{
    XGCValues gcv;
    Pixmap pixmap;
    Pixel fg, bg;

```

```

Window window = XtWindow(wid);
Display * display = XtDisplay(wid);
char arr[80];
char cname[20], temp[9];
struct file_node *count temp;
XFontStruct * font_info;
int x1, y1, x2, y2, i, centrex, centrey, topx, topy, count, up_count,
down_count;
XtVaGetValues(wid, XmNforeground, &fg, XmNbackground, &bg, NULL);

XClearWindow(display, window);
gcv.foreground = BlackPixelOfScreen(XtScreen(wid));
gc = XCreateGC(display, RootWindowOfScreen(XtScreen(wid)), GCForeground, &gcv);
XSetForeground(display, gc, WhitePixelOfScreen(XtScreen(wid)));
XFillRectangle(display, z_in_pixmap, gc, 0, 0, 700, 600);
XSetForeground(display, gc, BlackPixelOfScreen(XtScreen(wid)));
centrex = 300;
centrey = 250;
x2 = 100;
y2 = 100;

if ((font_info = XLoadQueryFont(display,
    "-adobe-courier-bold-r-*-*-20-*-*-*-*-*")
    == NULL)
    ;
else
    XSetFont(display, gc, font_info->fid);
if (cur) {
    /* Current node is present, display its information */
    XDrawString(XtDisplay(wid), XtWindow(wid), gc, 5, 580, "Press F1 key for
        help", 21);
    XDrawString(XtDisplay(wid), z_in_pixmap, gc, 5, 580, "Press F1 key for
        help", 21);

    /* All the drawings are made in the drawing widget as well as in a pixmap.
     * This pixmap is redrawn when the drawing area widget is exposed.
     */
    XDrawRectangle(XtDisplay(wid), XtWindow(wid), gc, centrex, centrey, x2,
        y2);
    XDrawRectangle(XtDisplay(wid), z_in_pixmap, gc, centrex, centrey, x2, y2);
    XDrawLine(XtDisplay(wid), XtWindow(wid), gc, centrex + x2 * 0.7, centrey,
        centrex + x2 * 0.7, centrey + y2);

    /* Shade the current node with 25_foreground pixmap. */
    pixmap = XmGetPixmap(XtScreen(wid), "25_foreground" , bg, fg);
    XSetStipple(display, gc, pixmap);
    XSetFillStyle(display, gc, FillStippled);
    XFillRectangle(display, window, gc, centrex, centrey, x2, y2);
    XFillRectangle(display, z_in_pixmap, gc, centrex, centrey, x2, y2);
    XSetFillStyle(display, gc, FillSolid);
    memset(temp, '\0', 9);
    if (strlen(cur->name) > 8)
        strncpy(temp, cur->name, 8);
    else
        strcpy(temp, cur->name);
    strcat(temp, "\0");
    XDrawString(XtDisplay(wid), XtWindow(wid), gc, centrex + 10, centrey + 50,
        temp, 7);
    XDrawString(XtDisplay(wid), z_in_pixmap, gc, centrex + 10, centrey + 50,
        temp, 7);

    /* If the status of current node is Experimental display it
     * after changing the font.
     */
    if (!cur->status) {
        if ((font_info = XLoadQueryFont(display,
            "-adobe-times-medium-i-*-*-17-*-*-*-*-*")
            == NULL)
            ;
        else
            XSetFont(display, gc, font_info->fid);
        XDrawString(XtDisplay(wid), XtWindow(wid), gc, centrex + 5, centrey

```

```

        + 80, "Experimental", 12);
XDrawString(XtDisplay(wid), z_in_pixmap, gc, centrex + 5, centrey +
80, "Experimental", 12);
if ((font_info = XLoadQueryFont(display,
"-adobe-courier-bold-r-*-*20-*-*-*-*")
== NULL)
;
else
XSetFont(display, gc, font_info->fid);
}
count = 0;

/* Check to see if the upward pointing arrow is necessary. */

up_count = 0;
down_count = 0;
count_temp = cur->left;
if (cur->left && child) {
    while (count_temp != child) {
        count_temp = count_temp->right;
        up_count++;
    }
}

/* Check to see if the downward pointing arrow is necessary. */

count_temp = child;
if (count_temp && count_temp->right) {
    count_temp = count_temp->right->right;
    while (count_temp && count_temp->right) {
        down_count++;
        count_temp = count_temp->right;
    }
}
if (up_count) {
    /* The upward pointing arrow is necessary and there are
    * up_count number of children in that direction.
    */

    sprintf(temp, "%d", up_count);
    XDrawString(XtDisplay(wid), XtWindow(wid), gc, 390 - strlen(temp) *
8, 275, temp, strlen(temp));
    XDrawString(XtDisplay(wid), z_in_pixmap, gc, 390 - strlen(temp) *
8, 275, temp, strlen(temp));
    draw_arrow(390, 250, TOP, 1, gc, XtDisplay(wid), XtWindow(wid));
    draw_arrow(390, 250, TOP, 1, gc, XtDisplay(wid), z_in_pixmap);
}
if (down_count) {
    /* The downward pointing arrow is necessary and there are
    * down_count number of children in that direction.
    */

    sprintf(temp, "%d", down_count);
    XDrawString(XtDisplay(wid), XtWindow(wid), gc, 390 - strlen(temp) *
8, 330, temp, strlen(temp));
    XDrawString(XtDisplay(wid), z_in_pixmap, gc, 390 - strlen(temp) *
8, 330, temp, strlen(temp));
    draw_arrow(390, 350, BOTTOM, 1, gc, XtDisplay(wid), XtWindow(wid));
    draw_arrow(390, 350, BOTTOM, 1, gc, XtDisplay(wid), z_in_pixmap);
}

/* Check to see if parent of current node is present. If so,
* display it.
*/

topx = 100;
topy = 250;
top = cur;

/* Because of the mapping of a tree with n children to a binary
* tree (refer to load.c for details ) the parent of a node
* in the binary tree may not be its parent in the n child tree.
* For example

```

```

*
*      a
*     / \
*    b   c
*   / \ / \
*  d  e f  g
*   / \
*  h  i
*

```

* The parent of d is b. The parent of i is also b and not d.

* The following loop locates the parent of a given node.

*/

```

while (top->parent && strcmp(top->parent->left->name, top->name) != 0)
    top = top->parent;
if (top->parent)
    top = top->parent;
if (top != cur) {
    if (top->name[0] != '\0') {

        /* Parent is present, display it. */

        XDrawRectangle(XtDisplay(wid), XtWindow(wid), gc, topx,
            topy, x2, y2);
        XDrawLine(XtDisplay(wid), XtWindow(wid), gc, topx + x2, topy
            + y2 / 2, centrex, centrey + y2 / 2);
        XDrawRectangle(XtDisplay(wid), z_in_pixmap, gc, topx, topy,
            x2, y2);
        XDrawLine(XtDisplay(wid), z_in_pixmap, gc, topx + x2, topy +
            y2 / 2, centrex, centrey + y2 / 2);
        memset(temp, '\0', 9);
        if (strlen(top->name) > 8)
            strncpy(temp, top->name, 8);
        else
            strcpy(temp, top->name);
        strcat(temp, "\0");
        XDrawString(XtDisplay(wid), XtWindow(wid), gc, topx + (x2 -
            strlen(temp) * 8) / 2, topy + 50, temp, 7);
        XDrawString(XtDisplay(wid), z_in_pixmap, gc, topx + (x2 -
            strlen(temp) * 8) / 2, topy + 50, temp, 7);

        /* If the status of the parent is Experimental display it.

        if (!top->status) {
            if ((font_info = XLoadQueryFont(display,
                "-adobe-times-medium-i-*-17-*-*-*-*")
                == NULL)
                ;
            else
                XSetFont(display, gc, font_info->fid);
            XDrawString(XtDisplay(wid), XtWindow(wid), gc, topx
                + 5, topy + 80, "Experimental", 12);
            XDrawString(XtDisplay(wid), z_in_pixmap, gc, topx +
                5, topy + 80, "Experimental", 12);
            if ((font_info = XLoadQueryFont(display,
                "-adobe-courier-bold-r-*-20-*-*-*-*")
                == NULL)
                ;
            else
                XSetFont(display, gc, font_info->fid);

        }

    }

x1 = 500;
y1 = 50;
x2 = 100;
y2 = 100;

/* Child is a global variable pointing to the node which is to be
 * displayed as the first child of the current node. This variable
 * is changed each time the user pushes the arrow.
 * In this way the scrolling effect is achieved.
 */

if (child) {
    childold = child;

```



```

/* Display three children from Child, if present */
for (i = 0; (i < 3) && child; i++) {
    XDrawRectangle(XtDisplay(wid), XtWindow(wid), gc, x1, y1,
                  x2, y2);
    XDrawLine(XtDisplay(wid), XtWindow(wid), gc, x1, y1 + y2 *
              0.5, centrex + x2, centrey + y2 * 0.5);
    XDrawRectangle(XtDisplay(wid), z_in_pixmap, gc, x1, y1, x2,
                  y2);
    XDrawLine(XtDisplay(wid), z_in_pixmap, gc, x1, y1 + y2 *
              0.5, centrex + x2, centrey + y2 * 0.5);
    memset(temp, '\0', 9);
    if (strlen(child->name) > 8)
        strncpy(temp, child->name, 8);
    else
        strcpy(temp, child->name);
    strcat(temp, "\0");
    XDrawString(XtDisplay(wid), XtWindow(wid), gc, x1 + (x2 -
                  strlen(temp) * 8) / 2, y1 + y2 / 2, temp, 7);
    XDrawString(XtDisplay(wid), z_in_pixmap, gc, x1 + (x2 -
                  strlen(temp) * 8) / 2, y1 + y2 / 2, temp, 7);

    /* If the status of the child is Experimental, display it */
    if (!child->status) {
        if ((font_info = XLoadQueryFont(display,
            "-adobe-times-medium-i-*-17-*-*-*-*-*")
            == NULL)
            ;
        else
            XSetFont(display, gc, font_info->fid);
        XDrawString(XtDisplay(wid), XtWindow(wid), gc, x1 +
                    5, y1 + y2 * 0.8, "Experimental", 12);
        XDrawString(XtDisplay(wid), z_in_pixmap, gc, x1 + 5,
                    y1 + y2 * 0.8, "Experimental", 12);
        if ((font_info = XLoadQueryFont(display,
            "-adobe-courier-bold-r-*-20-*-*-*-*-*")
            == NULL)
            ;
        else
            XSetFont(display, gc, font_info->fid);
    }
    y1 += 200;
    child = child->right;
}

/* Childnew is set to the last child displayed. */
if (i == 3)
    childnew = childold->right->right;
if (i == 2)
    childnew = childold->right;
if (i == 1)
    childnew = childold;
child = childold;
}
XtFree(font_info);
}

/* This function handles all the user interaction in the drawing area widget
 * in the zoom_in mode. In the zoom_out mode no interaction is expected in this
 * area. The user is expected to interact only with the Pushbuttons which are
 * handled by their callback functions.
 */

void draw_callback(Widget wid, XtPointer data, XmDrawingAreaCallbackStruct *cbs)
{
    int    x, y;
    char  str[50];
    struct list *p;
    XEvent * event = cbs->event;
    memset(str, '\0', 50);
    if (mode == z_in) {
        if (cbs->reason == XmCR_INPUT) {
            if (event->type == ButtonPress) {
                x = event->xbutton.x;

```

```

y = event->xbutton.y;

/* Check if the user clicked on the current node. */
if (x > 300 && x < 400) {
    if (y > 250 && y < 350)
        strcpy(str, cur->name);
}

/* Check if the user clicked on the top arrow of the current
 * node.
 * If so, move the child pointer two nodes up or as much
 * as possible within two nodes.
 */
if (x > 380 && x < 400) {
    if (y > 250 && y < 300) {
        if (child) {
            if (child->parent && child->parent
                != cur) {
                if (child->parent->parent &&
                    child->parent->parent !=
                    cur) {
                    childnew =
                        child->parent->parent;
                } else
                    childnew =
                        child->parent;
                child = childnew;
            }
        }
        treel(wid);
        return;
    }
}

/* Check if the user clicked on the down arrow of the
 * current node. If so, move the child to three nodes down
 * or as much as possible within three nodes.
 */
if (x > 380 && x < 400) {
    if (y >= 300 && y < 350) {
        if (child) {
            if (childnew) {
                if (childnew->right) {
                    child =
                        childnew->right;
                    if (childnew->right->right) {
                        child =
                            childnew->right->right;
                    } else
                        child =
                            childnew->right;
                }
            }
            treel(wid);
            return;
        }
    }
}

/* Check if the user clicked on the parent of the current
 * node.
 * If the parent is present, make it the current node and
 * redisplay.
 */
if (x > 100 && x < 200) {
    if (y > 250 && y < 350) {
        if (top && top != cur)
            strcpy(str, top->name);
    }
}

/* Check if the user clicked on the first displayed child of
 * the current node. If so, make it the current node if

```

```

    * present and redisplay.
    */

if (x > 500 && x < 600) {
    if (y > 50 && y < 150) {
        if (child)
            strcpy(str, child->name);
    }
}

/* Check if the user clicked on the second displayed child
 * of the current node. If so, make it the current node if
 * present and redisplay.
 */

if (x > 500 && x < 600) {
if (x > 500 && x < 600) {
    if (y > 250 && y < 350) {
        if (child) {
            if (child->right)
                strcpy(str,
                    child->right->name);
        }
    }
}

/* Check if the user clicked on the third displayed child of
 * the current node. If so, make it the current node if
 * present and redisplay.
 */

if (x > 500 && x < 600) {
if (x > 500 && x < 600) {
    if (y > 450 && y < 550) {
        if (child) {
            if (child->right) {
                if (child->right->right)
                    strcpy(str,
                        child->right->right->name);
            }
        }
    }
}

}

if (str[0] != '\0') {
    point = NULL;
    traverse(head, str);
    if (point) {
        cur = point;
        child = point->left;
        treel(wid);
    } else
        errorDialog("FILE NOT FOUND IN
            LIBRARY.....");
}

}
}

}

/* This function is used to draw the various arrows in the drawing area widget. */
void draw_arrow(int cordx, int cordy, int dirn, int type, GC myGC, Display*display,
Window window)
{
    XPoint popoints[5];
    switch (dirn) {
    case RIGHT :
        popoints[0].y = cordy, popoints[0].x = cordx, popoints[1].y = cordy +
            25, popoints[1].x = cordx - 50, popoints[2].y = cordy,
        popoints[2].x = cordx - 50, popoints[3].y = cordy - 25, popoints[3].x =
        cordx - 50, popoints[4].y = cordy, popoints[4].x = cordx;
        break;
    case REQLEFT :
        popoints[0].y = cordy + 7, popoints[0].x = cordx - 12, popoints[1].y =

```

```

    cordy, popoints[1].x = cordx, popoints[2].y = cordy - 7, popoints[2].x =
    cordx - 12 , popoints[3].y = cordy, popoints[3].x = cordx - 12;
    break;

case TOP :
    popoints[0].y = cordy + 10, popoints[0].x = cordx - 8 , popoints[1].y =
    cordy, popoints[1].x = cordx, popoints[2].y = cordy + 10, popoints[2].x =
    cordx + 8 , popoints[3].y = cordy + 10, popoints[3].x = cordx ;
    break;
case LEFT :
    popoints[0].y = cordy, popoints[0].x = cordx , popoints[1].y = cordy +
    25, popoints[1].x = cordx + 50, popoints[2].y = cordy, popoints[2].x =
    cordx + 50 , popoints[3].y = cordy - 25, popoints[3].x = cordx + 50;
    popoints[4].y = cordy, popoints[4].x = cordx;
    break;
case REQRIGHT :
    popoints[0].y = cordy + 7, popoints[0].x = cordx + 12 , popoints[1].y =
    cordy, popoints[1].x = cordx, popoints[2].y = cordy - 7, popoints[2].x =
    cordx + 12 , popoints[3].y = cordy, popoints[3].x = cordx + 12;
    break;
case BOTTOM :
    popoints[0].y = cordy - 10, popoints[0].x = cordx - 8, popoints[1].y =
    cordy , popoints[1].x = cordx, popoints[2].y = cordy - 10, popoints[2].x =
    cordx + 8 , popoints[3].y = cordy - 10, popoints[3].x = cordx ;
}
if (type)
    XFillPolygon(display, window, myGC, popoints, 4, Convex, CoordModeOrigin);
else
    XDrawLines(display, window, myGC, popoints, 5, CoordModeOrigin);
XSynchronize(display, TRUE);

return;
}

/* This function handles the zoom out mode of the display. The various points to
 * be displayed are stored in a s_tstructure. This function calculates which node is
 * to be displayed at each point. Once the array of the structure is filled up,
 * the array is displayed using the display_tree function.
 */

```

```

void zoom_out(Widget w, XtPointer wid, XmPushButtonCallbackStruct *cbs)
{
    struct file_node *temp;
    int i;
    initialize();
    if (!zoom_end) {
        if (right_pointer == NULL)
            right_pointer = cur;
        mode = z_out;
        mydisplay[0][2].point = cur;
        temp = mydisplay[0][2].point;

        /* Fill the array of structure horizontally (along the middle) indicating
         * the path of the current node from the parent.
         */

        if (temp->parent && temp->parent->left != temp) {
            mydisplay[0][1].point = temp->parent;
            if (temp->parent->parent && temp->parent->parent->left !=
                temp->parent)
                mydisplay[0][0].point = temp->parent->parent;
        }
        if (temp->right) {
            mydisplay[0][3].point = temp->right;
            if (temp->right->right) {
                mydisplay[0][4].point = temp->right->right;
            }
        }
        if (mydisplay[0][2].point != head) {
            mydisplay[0][5].point = parent(mydisplay[0][2].point);
            mydisplay[0][5].point = mydisplay[0][5].point->left;
        } else
            mydisplay[0][5].point = NULL;

        /* Fill the array for the other nodes. */

        temp = parent(right_pointer);
        i = 1;
    }
}

```

```

while (temp && i < 6) {
    mydisplay[i][2].point = temp;
    if (temp->parent && temp->parent->left != temp) {
        mydisplay[i][1].point = temp->parent;
        if (temp->parent->parent && temp->parent->parent->left !=
            temp->parent)
            mydisplay[i][0].point = temp->parent->parent;
    }
    if (temp->right) {
        mydisplay[i][3].point = temp->right;
        if (temp->right->right) {
            mydisplay[i][4].point = temp->right->right;
        }
    }
    if (mydisplay[i][2].point != head) {
        mydisplay[i][5].point = parent(mydisplay[i][2].point);
        mydisplay[i][5].point = mydisplay[i][5].point->left;
    } else
        mydisplay[i][5].point = NULL;
    temp = parent(temp);
    i++;
}

/* Load other information of the nodes. */

for (i = 0; i < 6; i++)
    load_depth(i);

/* Calculate if the arrow at left indicating more nodes to be seen
 * is necessary and calculate the number to be displayed in it if
 * necessary.
 */

if (parent(mydisplay[5][2].point) != NULL) {
    right_pointer = mydisplay[5][2].point;
    temp = right_pointer;
    left_level = 0;
    while ((temp = parent(temp)) != NULL)
        left_level++;
} else {
    right_pointer = NULL;
    zoom_end = 1;
    left_level = 0;
}
display_tree(wid);
} else
    errordialog("Cannot zoom out anymore .... ");
}

/* This function is called to initialize the variables for smooth transition
 * between the zoom_in and the zoom_out modes. */

void initialize()
{
    int i, j;
    for (i = 0; i < 6; i++) {
        for (j = 0; j < 6; j++) {
            mydisplay[i][j].point = NULL;
            mydisplay[i][j].sibling_no = 0;
            mydisplay[i][j].depth = 0;
            mydisplay[i][j].nodes = 0;
            if (i == 0)
                mypoint[i][j].x = 700 - i * 100;
            else
                mypoint[i][j].x = 600 - i * 100;
            mypoint[i][j].y = 50 + 100 * j;
        }
    }
    mypoint[6][2].x = 600;
    mypoint[6][2].y = 250;
    mypoint[7][2].x = 10;
    mypoint[7][2].y = 250;
}

```

```

/* This function can be used to find the parent of any given node.
 * As explained above in the zoom_in mode, the parent of a node in the
 * binary tree may not be the parent in the n children tree.
 */
struct file_node *parent(struct file_node *child)
{
    if (child) {
        while (child && child->parent->left != child)
            child = child->parent;
        if (child)
            return(child->parent);
        else
            return(NULL);
    } else
        return(NULL);
}

/* This function calculates the number of siblings of any given node. */
int sibling(struct file_node *child)
{
    struct file_node *my_temp;
    int count = 0;

    if (child) {
        my_temp = parent(child);
        if (my_temp) {
            my_temp = my_temp->left;
            while (my_temp) {
                my_temp = my_temp->right;
                count++;
            }
        }
    }
    return(count);
}

/* This function displays the filled array of the function zoom_out on the display.
 * Each node is displayed on the screen at the points dictated by the Mypoints array
 * of the structure.
 */
void display_tree(Widget wid)
{
    GC gc;
    XGCValues gcv;
    struct file_node *temp;
    Pixel fg, bg;
    Pixmap pixmap, pixmap1, pixmap2;
    XFontStruct * font_info;
    char str[80];
    int x, y, j, xprev, top, i, sib, yleft, c, xl;
    Window window = XtWindow(wid);
    Display * display = XtDisplay(wid);

    gcv.foreground = BlackPixelOfScreen(XtScreen(wid));
    gc = XCreateGC(display, RootWindowOfScreen(XtScreen(wid)), GCForeground, &gcv);
    XClearWindow(display, window);
    XtVaGetValues(wid, XmNforeground, &fg, XmNbackground, &bg, NULL);
    sprintf(str, "%snode", lib_d);
    pixmap = XmGetPixmap(XtScreen(wid), "25_foreground", bg, fg);
    pixmap1 = XmGetPixmap(XtScreen(wid), "75_foreground", bg, fg);

    XSetForeground(display, gc, WhitePixelOfScreen(XtScreen(wid)));
    XFillRectangle(display, window, pixmap, gc, 0, 0, 700, 600);
    XSetForeground(display, gc, BlackPixelOfScreen(XtScreen(wid)));
    if ((font_info = XLoadQueryFont(display,
        "-adobe-courier-bold-r-normal-*-14-100-*-*-*-*")
        == NULL)
        ;
    else
        XSetFont(display, gc, font_info->fid);

    XDrawString(XtDisplay(wid), XtWindow(wid), gc, 5, 580, "Press F1 key for help",
        21);
}

```

```

XDrawString(XtDisplay(wid), z_in_pixmap, gc, 5, 580, "Press F1 key for help", 21);
for (i = 0; i < 6; i++) {
    for (j = 0; j < 6; j++) {
        if (mydisplay[i][j].point) {
            /* It was initially intended to have three shapes for a given node.
            * 0 -> single node -> filled circle.
            * 1 -> normal node -> filled rectangle with depth and no of
            *   children.
            * 2 -> Info node -> rectangle with no of nodes in unseen
            *   children and the max depth in unseen children.
            *   This is displayed at bottom if necessary.
            * These shapes are in addition to the special shapes for head
            * and current node.
            *
            * case 0 was later discontinued to lessen the complexity
            * of understanding the display.
            * If case 0 needs to be reintroduced, uncomment the shape
            * lines in load_depth function.
            */

            switch (mydisplay[i][j].shape) {
            case 0 :
                if (mydisplay[i][j].point == cur) {

                    /* Single node, hence shape is filled circle. */

                    XDrawArc(display, window, gc, mypoint[i][j].x
                        - 10, mypoint[i][j].y - 10, diameter
                        + 20, diameter + 20, 360 * 64, 360 *
                        64);
                    XDrawArc(display, z_out_pixmap, gc,
                        mypoint[i][j].x - 10,
                        mypoint[i][j].y - 10, diameter + 20,
                        diameter + 20, 360 * 64, 360 * 64);
                    XSetStipple(display, gc, pixmap1);
                    XSetFillStyle(display, gc, FillStippled);
                    XFillArc(display, window, gc, mypoint[i][j].x
                        - 10, mypoint[i][j].y - 10, diameter + 20,
                        diameter + 20, 360 * 64, 360 * 64);
                    XFillArc(display, z_out_pixmap, gc,
                        mypoint[i][j].x - 10, mypoint[i][j].y - 10,
                        diameter + 20, diameter + 20, 360 * 64, 360 *
                        64);
                    XSetFillStyle(display, gc, FillSolid);
                    XSetForeground(display, gc,
                        WhitePixelOfScreen(XtScreen(wid)));
                    XFillArc(display, window, gc,
                        mypoint[i][j].x, mypoint[i][j].y, diameter,
                        diameter, 360 * 64, 360 * 64);
                    XFillArc(display, z_out_pixmap, gc,
                        mypoint[i][j].x, mypoint[i][j].y, diameter,
                        diameter, 360 * 64, 360 * 64);
                    XSetForeground(display, gc,
                        BlackPixelOfScreen(XtScreen(wid)));
                }
            if (mydisplay[i][j].point == head) {

                /* Single node and head, special shape. */

                XDrawArc(display, window, gc, mypoint[i][j].x
                    - 10, mypoint[i][j].y - 10, diameter + 20,
                    diameter + 20, 360 * 64, 360 * 64);
                XDrawArc(display, z_out_pixmap, gc,
                    mypoint[i][j].x - 10, mypoint[i][j].y - 10,
                    diameter + 20, diameter + 20, 360 * 64, 360
                    * 64);
            }
            XDrawArc(display, window, gc, mypoint[i][j].x,
                mypoint[i][j].y,
                diameter, diameter, 360 * 64, 360 * 64);
            XDrawArc(display, z_out_pixmap, gc, mypoint[i][j].x
                , mypoint[i][j].y, diameter, diameter, 360 * 64,
                360*64); XSetStipple(display, gc, pixmap);
            XSetFillStyle(display, gc, FillStippled);
            XFillArc(display, window, gc, mypoint[i][j].x,
                mypoint[i][j].y,

```

```

        diameter, diameter, 360 * 64, 360 * 64);
XFillArc(display, z_out_pixmap, gc, mypoint[i][j].x,
mypoint[i][j].y, diameter, diameter, 360 * 64,
360*64);
XSetFillStyle(display, gc, FillSolid);
break;
case 1:
if (mydisplay[i][j].point == cur) {

    /* Regular filled rectangle shape. */

    XDrawRectangle(display, window, gc,
        mypoint[i][j].x - 10, mypoint[i][j].y - 10,
        70, 50);
    XDrawRectangle(display, z_out_pixmap, gc,
        mypoint[i][j].x - 10, mypoint[i][j].y -
        10, 70, 50);
    XSetStipple(display, gc, pixmap1);
    XSetFillStyle(display, gc, FillStippled);
    XFillRectangle(display, window, gc,
        mypoint[i][j].x - 10, mypoint[i][j].y - 10,
        70, 50);
    XFillRectangle(display, z_out_pixmap, gc,
        mypoint[i][j].x - 10, mypoint[i][j].y - 10,
        70, 50);
    XSetFillStyle(display, gc, FillSolid);
    XSetForeground(display, gc,
        WhitePixelOfScreen(XtScreen(wid)));
    XFillRectangle(display, window, gc,
        mypoint[i][j].x, mypoint[i][j].y, 50, 30);
    XFillRectangle(display, z_out_pixmap, gc,
        mypoint[i][j].x, mypoint[i][j].y, 50,
        30);
    XSetForeground(display, gc,
        BlackPixelOfScreen(XtScreen(wid)));
}
if (mydisplay[i][j].point == head) {

    /* Node is head, special shape. */

    XDrawRectangle(display, window, gc,
        mypoint[i][j].x - 10, mypoint[i][j].y - 10,
        70, 50);
    XDrawRectangle(display, z_out_pixmap, gc,
        mypoint[i][j].x - 10, mypoint[i][j].y - 10,
        70, 50);
}
XDrawRectangle(display, window, gc, mypoint[i][j].x,
mypoint[i][j].y, 50, 30);
XDrawRectangle(display, z_out_pixmap, gc,
mypoint[i][j].x, mypoint[i][j].y, 50, 30);
XSetStipple(display, gc, pixmap);
XSetFillStyle(display, gc, FillStippled);
XFillRectangle(display, window, gc, mypoint[i][j].x,
mypoint[i][j].y, 50, 30);
XFillRectangle(display, z_out_pixmap, gc,
mypoint[i][j].x, mypoint[i][j].y, 50, 30);
XSetFillStyle(display, gc, FillSolid);
sprintf(str, "%d,%d", mydisplay[i][j].nodes,
mydisplay[i][j].depth);
XDrawString(display, window, gc, mypoint[i][j].x +
3, mypoint[i][j].y + 20, str, strlen(str));
XDrawString(display, z_out_pixmap, gc,
mypoint[i][j].x + 3, mypoint[i][j].y + 20, str,
strlen(str));

break;
case 2:
if (mydisplay[i][j].nodes != 0) {

    /* Hidden siblings, display information node. */

    XDrawRectangle(display, window, gc,
        mypoint[i][j].x, mypoint[i][j].y, 50, 30);
    XDrawRectangle(display, z_out_pixmap, gc,
        mypoint[i][j].x,
        mypoint[i][j].y, 50, 30);
    sprintf(str, "%d,%d", mydisplay[i][j].nodes,

```



```
        mydisplay[i][j].depth);
    XDrawString(display, window, gc,
        mypoint[i][j].x + 3, mypoint[i][j].y + 20,
        str, strlen(str));
    XDrawString(display, z_out_pixmap, gc,
        mypoint[i][j].x + 3, mypoint[i][j].y + 20,
        str, strlen(str));
}
}
/* If left triangle, handle the connections as a special
   case. */
if (i == 5 && left_level) {
    x = mypoint[7][2].x;
    y = mypoint[7][2].y;
    x1 = 50;
    if (j != 5) {
        XDrawLine(display, window, gc, x + x1, y +
            15, mypoint[i][j].x, mypoint[i][j].y + 15);
        XDrawLine(display, z_out_pixmap, gc, x + x1,
            y + 15, mypoint[i][j].x, mypoint[i][j].y +
            15);
    } else {
        if (mydisplay[i][j].nodes != 0) {
            XDrawLine(display, window, gc, x +
                x1, y + 15, mypoint[i][j].x,
                mypoint[i][j].y + 15);
            XDrawLine(display, z_out_pixmap, gc,
                x + x1, y + 15, mypoint[i][j].x,
                mypoint[i][j].y + 15);
        }
    }
}
if (i != 5) {
    if (mydisplay[i+1][2].point) {
        x = mypoint[i+1][2].x;
        y = mypoint[i+1][2].y;
        x1 = 50;

        /* The following code is commented as right
           * triangle to display the hidden levels has
           * been discontinued.
           */

        /*
           if(i==0 && right_level)
           {
               x=mypoint[6][2].x;
               y=mypoint[6][2].y;
               x1=50;
           }

           */
        if (j != 5) {
            XDrawLine(display, window, gc, x +
                x1, y + 15, mypoint[i][j].x,
                mypoint[i][j].y + 15);
            XDrawLine(display, z_out_pixmap, gc,
                x + x1, y + 15, mypoint[i][j].x,
                mypoint[i][j].y + 15);
        } else {
            if (mydisplay[i][j].nodes != 0) {
                XDrawLine(display, window,
                    gc, x + x1, y + 15,
                    mypoint[i][j].x, mypoint[i][j].y
                    + 15);
                XDrawLine(display,
                    z_out_pixmap, gc, x + x1, y
                    + 15, mypoint[i][j].x,
                    mypoint[i][j].y + 15);
            }
        }
    }
}
}
}
```

```

    }
    x = mypoint[6][2].x + 50;
    y = mypoint[6][2].y + 15;

    /* The right triangle displaying the number of levels hidden is discontinued.
       Hence, the following code is commented.
    */

    /*
if(right_level)
{
    sprintf(str, "%d", right_level);
    draw_arrow(x, y, RIGHT, 0, gc, display, window);
    XDrawString(display, window, gc, x-45, y, str, strlen(str));
    XDrawLine(display, window, gc, x-50, y, mypoint[1][2].x+50, mypoint[1][2].y+15);
    draw_arrow(x, y, RIGHT, 0, gc, display, z_out_pixmap);
    XDrawString(display, z_out_pixmap, gc, x-45, y, str, strlen(str));
    XDrawLine(display, z_out_pixmap, gc, x-50, y, mypoint[1][2].x+50, mypoint[1][2].y+15);
}
*/

    x = mypoint[7][2].x;
    y = mypoint[7][2].y + 15;

    /* Display a triangle at the left to indicate the number of levels to the root
       * if necessary.
    */

    if (left_level) {
        sprintf(str, "%d", left_level);
        draw_arrow(x, y, LEFT, 0, gc, display, window);
        XDrawString(display, window, gc, x + 25, y, str, strlen(str));
        draw_arrow(x, y, LEFT, 0, gc, display, z_out_pixmap);
        XDrawString(display, z_out_pixmap, gc, x + 25, y, str, strlen(str));
    }

    if (parent(mydisplay[5][2].point))
        right_level += 5;
    XtFree(font_info);
}

/* This function is called when the zoom_in pushbutton is pushed. */
void zoom_in(Widget w, XtPointer wid, XmPushButtonCallbackStruct *cbs)
{
    zoom_end = 0;
    left_level = 0;
    right_level = 0;
    mode = z_in;
    right_pointer = NULL;
    tree1(wid);
}

void reset(Widget w, XtPointer wid, XmPushButtonCallbackStruct *cbs)
{
    zoom_end = 0;
    XtDestroyWidget(wid);
}

/* This function is called to redraw the drawing area widget when it is
   * exposed.
   */

void redraw(Widget wid, XtPointer data, XmDrawingAreaCallbackStruct *cbs)
{
    XEvent * event = cbs->event;
    Display * dpy = event->xexpose.display;
    if (!mode)
        XCopyArea(dpy, z_in_pixmap, cbs->window, gc, 0, 0, 700, 600, 0, 0);
    else
        XCopyArea(dpy, z_out_pixmap, cbs->window, gc, 0, 0, 700, 600, 0, 0);
}

/* This function is called to calculate the height of the given node. */
void depth(struct file_node *node)
{

```

```

if (node) {
    if ( node->parent && node->parent->left == node) {
        if (node != gl_node)
            curdepth++;
    }
    if (curdepth > maxdepth)
        maxdepth = curdepth;
    depth(node->left);
    if (node->left)
        curdepth--;
    if (node != gl_node)
        ;
    depth(node->right);
}
}

/* This function counts the number of nodes under a given node
 * including itself.
 */

void nodes(struct file_node *node)
{
    if (node) {
        node_count++;
        nodes(node->left);
        if (node != gl_node)
            nodes(node->right);
    }
}

/* This function is called to load the height and number of
 * nodes of all siblings at a time.
 */

void load_depth(int i)
{
    int j;
    for (j = 0; j < 5; j++) {
        if (mydisplay[i][j].point) {
            gl_node = mydisplay[i][j].point;
            node_count = 0;

            /* Count the nodes below. */

            nodes(mydisplay[i][j].point);
            mydisplay[i][j].nodes = node_count;

            /* The 0 shape has been discontinued
             * Hence the following lines are commented.
             * See display_tree function for more
             * information.
             */

            /* If node_count =1, set shape to 0
             * else set shape to 1.
             */

            /*
                if(node_count==1)
                    mydisplay[i][j].shape=0;
                else
                    mydisplay[i][j].shape=1;
            */
            mydisplay[i][j].shape = 1;

            gl_node = mydisplay[i][j].point;
            maxdepth = 0;
            curdepth = 0;

            /* Calculate the height of a node. */

            depth(mydisplay[i][j].point);
            mydisplay[i][j].depth = maxdepth;
        }
    }
    if (mydisplay[i][5].point) {

```

```

        /* Loading the information of the rectangle displayed
        * at the bottom.
        */

        m_depth(mydisplay[i][5].point, i);
        mydisplay[i][5].shape = 2;
        mydisplay[i][5].sibling_no = sibling(mydisplay[i][5].point);
        mydisplay[i][5].nodes = node_count;
        mydisplay[i][5].depth = maxdepth;
    }
}

/* This function is used to calculate the maximum height of all unseen nodes.
 * Traverse through all children, check if they are displayed. If not, calculate
 * the height. Get the maximum height of these nodes.
 */

void m_depth(struct file_node *node, int i)
{
    struct file_node *temp;
    int dep = 0;
    temp = node;
    node_count = 0;
    maxdepth = 0;
    while (temp) {
        if (test(temp, i)) {
            gl_node = temp;
            nodes(temp);
            gl_node = temp;
            maxdepth = 0;
            curdepth = 0;
            depth(temp);
            if (maxdepth > dep)
                dep = maxdepth;
        }
        temp = temp->right;
    }
    maxdepth = dep;
}

/* This function is used to test if a given node is on the display.
 * To do this, compare the node to all nodes in the structure.
 */

int test(struct file_node *node, int count)
{
    int i;
    for (i = 0; i < 5; i++) {
        if (node == mydisplay[count][i].point)
            return(0);
    }
    return(1);
}

/* This function is called when the legend pushbutton is pushed.
 * This function pops up a dialog with a drawing area where the
 * legend information is displayed. The dialog also has OK and cancel
 * pushbuttons to dismiss the dialog.
 */

/*

```

The structure of this widget is as follows.

```

/* Loading the information of the rectangle displayed
 * at the bottom.
 */

    m_depth(mydisplay[i][5].point, i);
    mydisplay[i][5].shape = 2;
    mydisplay[i][5].sibling_no = sibling(mydisplay[i][5].point);
    mydisplay[i][5].nodes = node_count;
    mydisplay[i][5].depth = maxdepth;
}

/* This function is used to calculate the maximum height of all unseen nodes.
 * Traverse through all children, check if they are displayed. If not, calculate
 * the height. Get the maximum height of these nodes.
 */

void m_depth(struct file_node *node, int i)
{
    struct file_node *temp;
    int dep = 0;
    temp = node;
    node_count = 0;
    maxdepth = 0;
    while (temp) {
        if (test(temp, i)) {
            gl_node = temp;
            nodes(temp);
            gl_node = temp;
            maxdepth = 0;
            curdepth = 0;
            depth(temp);
            if (maxdepth > dep)
                dep = maxdepth;
        }
        temp = temp->right;
    }
    maxdepth = dep;
}

/* This function is used to test if a given node is on the display.
 * To do this, compare the node to all nodes in the structure.
 */

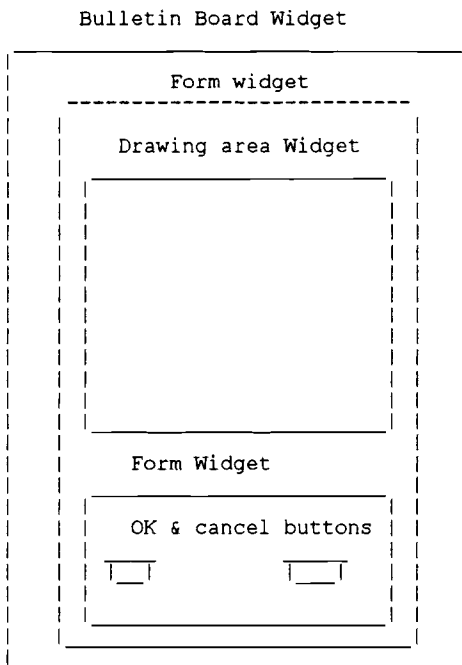
int test(struct file_node *node, int count)
{
    int i;
    for (i = 0; i < 5; i++) {
        if (node == mydisplay[count][i].point)
            return(0);
    }
    return(1);
}

/* This function is called when the legend pushbutton is pushed.
 * This function pops up a dialog with a drawing area where the
 * legend information is displayed. The dialog also has OK and cancel
 * pushbuttons to dismiss the dialog.
 */

/*

```

The structure of this widget is as follows.



*/

```

void draw_legend(Widget wid, XtPointer client_data, XmPushButtonCallbackStruct *cbs)
{
    Arg args[14];
    XmString one, two, three;
    Widget top, form1, ll, draw_w, pb1, pb2, pb3, rowcol3;
    int i;

    XtSetArg(args[0], XmNautoUnmanage, False);
    XtSetArg(args[1], XmNdefaultPosition, False);
    XtSetArg(args[2], XmNallowShellResize, True);
    XtSetArg(args[3], XmNwidth, 750);
    XtSetArg(args[4], XmNheight, 500);
    XtSetArg(args[5], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);

    top = XmCreateBulletinBoardDialog(toplevel, "LIST", args, 6);
    /* Position the dialog at 150,100. */
    XtAddCallback(top, XmNmapCallback, map, 150100);

    form1 = XtVaCreateManagedWidget("form", xmFormWidgetClass, top, NULL);

    ll = XtVaCreateManagedWidget("rowcol", xmFormWidgetClass, form1,
        XmNbottomAttachment, XmATTACH_FORM, XmNleftAttachment, XmATTACH_FORM,
        XmNrightAttachment, XmATTACH_FORM, XmNfractionBase, 5, NULL);
    draw_w = XtVaCreateManagedWidget("dbutton", xmDrawingAreaWidgetClass, form1,
        XmNtopAttachment, XmATTACH_FORM, XmNbottomAttachment, XmATTACH_WIDGET,
        XmNbottomWidget, ll, XmNwidth, 725, XmNheight, 400, XmNborderWidth, 2,
        XmNshadowThickness, 1, NULL);

    /* Call redraw to copy the pixmap to the drawing area Widget. */
    XtAddCallback(draw_w, XmNexposeCallback, rdraw, NULL);

    one = XmStringCreateSimple("OK");
    pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, ll, XmNlabelString,
        one, XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 1, NULL);
    XtFree(one);

    one = XmStringCreateSimple("Cancel");
    pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, ll, XmNlabelString,
        one, XmNleftAttachment, XmATTACH_POSITION, XmNleftPosition, 4, NULL);

```

```

XtFree(one);
XtAddCallback(pb1, XmNactivateCallback, destroy, top);
XtAddCallback(pb2, XmNactivateCallback, destroy, top);

XtManageChild(top);
}

/* This function is called to redraw the drawing area when it is exposed
 * Note that this function is also used to copy the pixmap onto the
 * drawing area for the first time also.
 */

void rdraw(Widget wid, XtPointer data, XmDrawingAreaCallbackStruct *cbs)
{
    GC gc;
    XGCValues gcv;
    Pixel fg, bg;
    Pixmap pixmap;
    Window window = XtWindow(wid);
    Display *display = XtDisplay(wid);
    char str[80];
    gcv.foreground = BlackPixelOfScreen(XtScreen(wid));
    gc = XCreateGC(display, RootWindowOfScreen(XtScreen(wid)), GCForeground, &gcv);
    XtVaGetValues(wid, XmNforeground, &fg, XmNbackground, &bg, NULL);
    sprintf(str, "%szoom", lib_d);
    pixmap = XmGetPixmap(XtScreen(wid), str, fg, bg);
    XCopyArea(display, pixmap, cbs->window, gc, 0, 0, 725, 400, 0, 0);
}
/*
* * * * *
*   Filename : Delete.c
* * * * *
*   Programmed by : N. Sunil Chakravarthy
* * * * *
*   Last updated on : 08.17.94
* * * * *

```

```

This file contains the code to accept the name of the file to be deleted.
*/

```

```

#include <Xm/DialogS.h>
#include <Xm/TextF.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/Form.h>
#include <Xm/SelectioB.h>
#include <Xm/PushB.h>
#include <Xm/LabelG.h>
#include <stdio.h>

extern destroy();
extern Destroy();
extern delete_node();
extern load();
extern map();
extern lock();
extern unlock();
extern help();

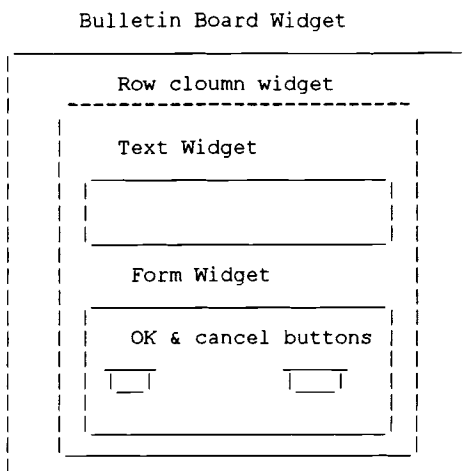
extern char lib_d[100];
extern char *passwd;
extern int load_time;
extern int sys_adm;
extern Widget toplevel;

void working_dialog();
void start_delete();

/* This function is called when the user pushes the delete drawn button
 * This function pops up a dialog with a text widget to receive the name
 * of the file to be deleted.
 */

/*
The structure of this widget is:

```



*/

```

void delete(Widget w)
{
    Widget      top, pb1, pb2, rowcol3, labell, l1, p1;
    Arg  args[10];
    int   i = 0;
    XmString one, two;
    XtSetArg(args[i], XmNautoUnmanage, True);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNwidth, 400);
    i++;
    XtSetArg(args[i], XmNheight, 300);
    i++;
    XtSetArg(args[i], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);

    top = XmCreateBulletinBoardDialog(w, "LIST", args, 6);

    /* Position the dialog at 150,400. */

    XtAddCallback(top, XmNmapCallback, map, 150400);
    XtAddCallback(top, XmNhelpCallback, help, 14);

    l1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, top, XmNnumColumns,
        1, XmNorientation, XmVERTICAL, NULL);

    one = XmStringCreateSimple("Give the name of the File to be deleted:");
    labell = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
        one, NULL);

    XtFree(one);
    p1 = XtVaCreateWidget("text", xmTextWidgetClass, l1, NULL);
    XtAddCallback(p1, XmNactivateCallback, start_delete, p1);

    rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, l1, NULL);
    one = XmStringCreateSimple("OK");
    two = XmStringCreateSimple("CANCEL");
    pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNheight, 25, NULL);

    pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
        XmNrightOffset, 5, XmNheight, 25, NULL);
    XtAddCallback(pb1, XmNactivateCallback, start_delete, p1);
    XtAddCallback(pb2, XmNactivateCallback, destroy, top);
    XtManageChild(rowcol3);
  
```



```

XtManageChild(label1);
XtManageChild(p1);
XtManageChild(l1);
XtManageChild(top);

}

/* This function is called when the user pushes the OK button in the above
 * dialog. If the user is the library administrator, then the delete_node
 * function is called else a note is made in the update.dat file for the
 * library administrator.
 */

void start_delete(Widget tw, XtPointer w, XmTextVerifyCallbackStruct *cbs)
{
    char *text;
    int time;
    char str[80];
    FILE * fp;

    text = XmTextGetString(w);

    Destroy(XtParent(XtParent(w)));

    /* Function to delete. */
    if (sys_adm)
        delete_node(text);
    else {

        /* User is not the library administrator, so make a note
         * in the update.dat file.
         */

        sprintf(str, "%supdate.dat", lib_d);
        fp = fopen(str, "r+");
        if (!fp) {
            errordialog("UNABLE TO OPEN update file ....change is not
                recorded");
        } else {
            if (lock(fp)) {
                fseek(fp, 0, 2);
                sprintf(str, "DELETE FILE %s ,requested by %s\n",
                    text, passwd);
                fputs(str, fp);
                unlock(fp);
                fclose(fp);
            } else
                errordialog("UNABLE TO LOCK update file ....Operation is not
                    successful");
        }
    }
}

/*
 * * * * *
 *   Filename : Move.c
 * * * * *
 *   Programmed by : N.Sunil Chakravarthy.
 * * * * *
 *   Last updated on : 08.17.94
 * * * * *
 */

This file contains the code to pop up the dialog to receive the file name
to be moved and the name of the new parent.
*/

#include <Xm/DialogS.h>
#include <Xm/TextF.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/Form.h>
#include <Xm/SelectioB.h>
#include <Xm/PushB.h>

```

```

#include <Xm/LabelG.h>
#include <stdio.h>

void start_move();
extern destroy();
extern Destroy();
extern help();
extern errorDialog();
extern map();
extern lock();
extern unlock();

extern char    lib_d[100];
extern char    *passwd;
extern int     sys_adm;
Widget p2;

/* This function is called when the move drawn button is pushed.
 * This function pops up a dialog with two text widgets to receive
 * the name of the file to be moved and the name of the new parent.
 */

void move(Widget w)
{
    Widget    top, pb1, pb2, rowcol3, labell, l1, p1, label2;
    Arg       args[10];
    int       i = 0;
    XmString  one, two;
    XtSetArg(args[i], XmNautoUnmanage, False);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNwidth, 400);
    i++;
    XtSetArg(args[i], XmNheight, 300);
    i++;
    XtSetArg(args[i], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);

    top = XmCreateBulletinBoardDialog(w, "LIST", args, 6);

    /* Position the dialog at 150,300. */

    XtAddCallback(top, XmNmapCallback, map, 150300);
    XtAddCallback(top, XmNhelpCallback, help, 13);

    l1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, top, XmNnumColumns,
        1, XmNorientation, XmVERTICAL, NULL);

    one = XmStringCreateSimple("Give the name of the File to be moved:");
    labell = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
        one, NULL);

    XtFree(one);
    p1 = XtVaCreateWidget("text", xmTextWidgetClass, l1, NULL);
    XtAddCallback(p1, XmNactivateCallback, XmProcessTraversal,
        XmTRAVERSE_NEXT_TAB_GROUP);

    one = XmStringCreateSimple("Give the name of the File to be attached to:");
    label2 = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
        one, NULL);

    XtFree(one);
    p2 = XtVaCreateWidget("text", xmTextWidgetClass, l1, NULL);
    XtAddCallback(p2, XmNactivateCallback, XmProcessTraversal,
        XmTRAVERSE_NEXT_TAB_GROUP);

    rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, l1, NULL);
    one = XmStringCreateSimple("OK");
    two = XmStringCreateSimple("CANCEL");
    pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNheight, 25, NULL);

    pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,

```

```

        XmNrightOffset, 5, XmNheight, 25, NULL);
XtAddCallback(pb1, XmNactivateCallback, start_move, p1);
XtAddCallback(pb2, XmNactivateCallback, destroy, top);
XtManageChild(rowcol3);
XtManageChild(label1);
XtManageChild(p1);
XtManageChild(label2);
XtManageChild(p2);
XtManageChild(l1);
XtManageChild(top);
}

/* This function is called when the user pushes the OK button in the above
 * dialog. If the user is the library administrator, then the move_node
 * function is called else a note is made in the update.dat file for the
 * library administrator.
 */

void start_move(Widget tw, XtPointer w, XmTextVerifyCallbackStruct *cbs)
{
    char *text, *text1;
    char str[100];
    FILE * fp;
    text = XmTextGetString(w);
    text1 = XmTextGetString(p2);

    XtUnmanageChild(XtParent(XtParent(w)));

    /* Function to move. */
    if (sys_adm)
        move_node(text, text1);
    else {

        /* User is not the library administrator, so make a note
         * in the update.dat file.
         */

        sprintf(str, "%supdate.dat", lib_d);
        fp = fopen(str, "r+");
        if (!fp) {
            errorDialog("UNABLE TO OPEN update file ...change not recorded");
        } else {
            if (lock(fp)) {
                fseek(fp, 0, 2);
                sprintf(str, "Move File %s to a child of %s requested by
                    %s\n",text, text1, passwd);
                fputs(str, fp);
                unlock(fp);
                fclose(fp);
            } else
                errorDialog("UNABLE TO LOCK update file ...OPERATION NOT
                    SUCCESSFUL");
        }
    }
}

/*
 * * * * *
 *   Filename : Options.c
 * * * * *
 *   Programmed by : N.Sunil Chakravarthy.
 * * * * *
 *   Last updated on : 08.17.94
 * * * * *
 */

This file has the code to pop up the dialog after the Options drawn button is
pushed.
*/

#include <Xm/DialogS.h>
#include <Xm/TextF.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>

```

```

#include <Xm/Form.h>
#include <Xm/SelectioB.h>
#include <Xm/PushB.h>
#include <Xm/LabelG.h>
# include <stdio.h>

void mode_set();
void util_ok();
void get_file();
void change_description();
void get_status_file();
void change_status();
void update_thesaurus();
void get_word();
void add_word();
void delete_word();
void eq_word();
int change_desc();

extern Widget rowcol;
extern Widget toplevel;
extern Widget name2, addr2, text_w1, func2, method2;

extern int sys_adm;
extern void destroy();
extern void Destroy();
extern void help();
extern int lock();
extern void unlock();
extern void list_display();
extern void system_select();
extern void add_node();
extern struct file_node {
    char name[20];
    char rcsfile[80];
    char rcsno[2000];
    char author[20];
    char email[20];
    char function[100];
    char method[100];
    char implementation[1000];
    float saved;
    int outno;
    int status;
    int filesize;
    struct file_node *left;
    struct file_node *right;
    struct file_node *parent;
};

extern struct file_node *new_file, *point, *head;
char file[100];

extern XmStringCharSet charset;
extern char dir[100], file_name[100], src_d[100], lib_d[100];

extern int gl_test;
extern char filename[80];

extern map();
extern toggle();

extern int mode;
extern Widget top, pb1, pb2, pb3, pb4, rowcol3, list_w, sb, label1,
    l1, p1;
char th_word[100];
Widget dialog;

/* This function is called when the Options drawn button is pushed.
 * This function pops up a dialog with two radiobuttons indicating two
 * choices: Change description and Add to thesaurus.
 */

void util(Widget w)
{
    Widget pb1, cin_R, cin_N, cin_S, cin_T, cin_M, rowcoll1, pb2, rowcol3,
        pb3, pb4, pb5, rad;

```

```

XmString one, two, three, four, five;
Arg args[10];
int i;
i = 0;
XtSetArg(args[0], XmNautoUnmanage, True);
XtSetArg(args[1], XmNdefaultPosition, False);
XtSetArg(args[2], XmNallowShellResize, True);
XtSetArg(args[3], XmNwidth, 200);
XtSetArg(args[4], XmNheight, 100);
XtSetArg(args[5], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);

dialog = XmCreateBulletinBoardDialog(w, "UTIL", args, 6);

/* Position the dialog at 150,400. */

XtAddCallback(dialog, XmNmapCallback, map, 150400);
XtAddCallback(dialog, XmNhelpCallback, help, 15);

rowcol1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, dialog,
                          XmNnumColumns, 1, XmNpacking, XmPACK_COLUMN, XmNorientation,
                          XmVERTICAL, NULL);
one = XmStringCreateSimple("Change description");
two = XmStringCreateSimple("Add to Thesaurus");
mode = 0;
rad = XmVaCreateSimpleRadioBox(rowcol1, "radio_box", 0, toggle, XmVaRADIOBUTTON,
                              one, 0, NULL, NULL, XmVaRADIOBUTTON, two, 0, NULL, NULL, NULL);
rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcol1,
                          NULL);

one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
                              XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
                              XmNbottomAttachment, XmATTACH_FORM, XmNleftOffset, 5,
                              XmNheight, 25, NULL);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
                              XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
                              XmNbottomAttachment, XmATTACH_FORM, XmNrightOffset,
                              5, XmNheight, 25, NULL);

XtAddCallback(pb1, XmNactivateCallback, util_ok, dialog);
XtAddCallback(pb2, XmNactivateCallback, destroy, dialog);
XmStringFree(one);
XmStringFree(two);
XtManageChild(rowcol1);
XtManageChild(rowcol3);
XtManageChild(rad);
XtManageChild(dialog);
}

/* This function is called when the radiobuttons are clicked.
 * mode is set to reflect which one was clicked.
 */

void mode_set(Widget w_id, int client_data, XmPushButtonCallbackStruct*cbs)
{
    mode = client_data;
}

/* This function is called when the OK button is pushed in the above dialog. */

void util_ok(Widget w_id, XtPointer client_data, XmPushButtonCallbackStruct*cbs)
{
    XtUnmanageChild(client_data);
    switch (mode) {

    case 0:

        /* Change description was selected. */

        get_file();
        break;

    case 1:

```

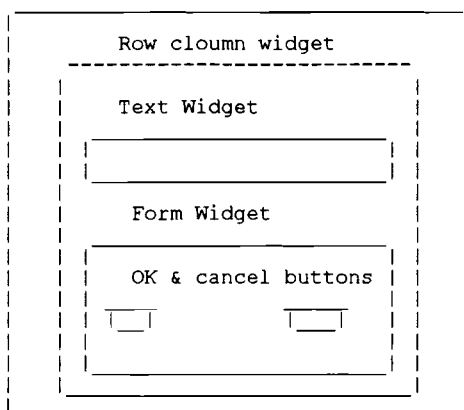
```

        /* Add to Thesaurus was selected. */
        get_word();
        break;
    }
}

/* This function is used to get the name of the file whose description
 * has to be changed.
 */
/*
    The structure of this widget is:

```

Bulletin Board Widget



```

*/

```

```

void get_file()
{
    Widget      top, pb1, pb2, rowcol3, labell, l1, p1;
    Arg  args[10];
    int   i = 0;
    XmString one, two;
    XtSetArg(args[0], XmNautoUnmanage, True);
    XtSetArg(args[1], XmNdefaultPosition, False);
    XtSetArg(args[2], XmNallowShellResize, True);
    XtSetArg(args[3], XmNwidth, 400);
    XtSetArg(args[4], XmNheight, 300);
    XtSetArg(args[5], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);

    top = XmCreateBulletinBoardDialog(toplevel, "LIST", args, 5);

    /* Position the dialog at 250,400. */

    XtAddCallback(top, XmNmapCallback, map, 250400);
    XtAddCallback(top, XmNhelpCallback, help, 19);

    l1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, top, XmNnumColumns,
        1, XmNorientation, XmVERTICAL, NULL);

    one = XmStringCreateSimple("Give the name of the File:");
    labell = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
        one, NULL);

    XtFree(one);
    p1 = XtVaCreateWidget("text", xmTextWidgetClass, l1, NULL);
    XtAddCallback(p1, XmNactivateCallback, change_description, p1);

    rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, l1, NULL);
    one = XmStringCreateSimple("OK");
    two = XmStringCreateSimple("CANCEL");

```

```

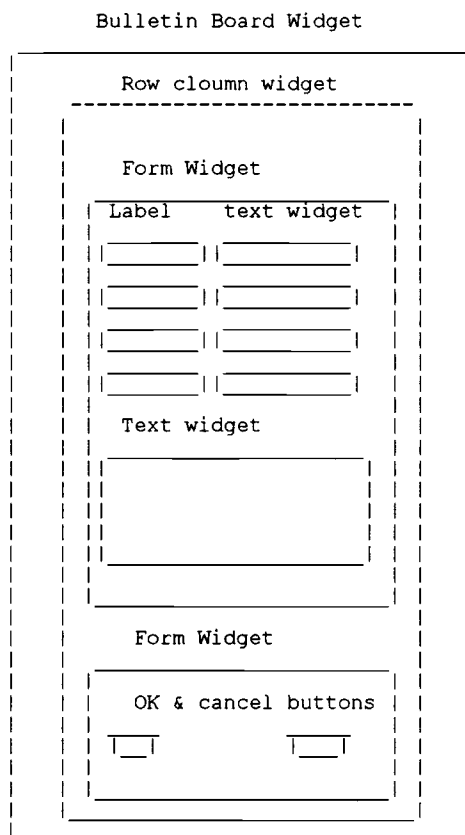
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
                               XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
                               XmNleftOffset, 5, XmNheight, 25, NULL);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
                               XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
                               XmNrightOffset, 5, XmNheight, 25, NULL);
XtAddCallback(pb1, XmNactivateCallback, change_description, p1);
XtAddCallback(pb2, XmNactivateCallback, destroy, top);
XtManageChild(rowcol3);
XtManageChild(label1);
XtManageChild(p1);
XtManageChild(l1);
XtManageChild(top);
}

/* This function is called after the user types the name of the file whose
 * description is to be changed and pushes the OK button. This function
 * locates the file and displays its description in a dialog so that the user
 * can change the description as needed.
 */

```

The structure of this dialog box is as follows.



*/

```

void change_description(Widget tw, XtPointer w, XmTextVerifyCallbackStruct *cbs)
{
    char *text;
    int pos;
    Widget label, label1, rowcol2, rowcol3, pb1, pb2, form1, name1, addr1,
          func1, method1;
    Widget dialog1;

    Arg args[10];

```

```

XmString one, two;
int i = 0;

/* Get the name from the text widget. */

text = XmTextGetString(w);
XtUnmanageChild(XtParent(XtParent(w)));

/* Locate the file. */

point = NULL;
strcpy(file, text);
traverse(head->left, text);
if (point) {

    /* File is located, display its description. */

    XtSetArg(ags[0], XmNautoUnmanage, True);
    XtSetArg(ags[1], XmNdefaultPosition, False);
    XtSetArg(ags[2], XmNallowShellResize, True);
    XtSetArg(ags[3], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);
    dialog1 = XmCreateBulletinBoardDialog(topLevel, "DESCRIPTION",
        ags, 3);

    /* Position the dialog at 350,400. */

    XtAddCallback(dialog1, XmNmapCallback, map, 350400);
    XtAddCallback(dialog1, XmNhelpCallback, help, 1);

    rowcol2 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass,
        dialog1, XmNnumColumns, 1, XmNorientation, XmVERTICAL, NULL);
    form1 = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcol2,
        NULL);

    one = XmStringCreateSimple("NAME OF THE AUTHOR");
    name1 = XtVaCreateWidget("label", xmLabelWidgetClass, form1,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNheight, 25, NULL);
    XtSetArg(ags[0], XmNrows, 1);
    XtSetArg(ags[1], XmNcolumns, 20);
    XtSetArg(ags[2], XmNleftAttachment, XmATTACH_WIDGET);
    XtSetArg(ags[3], XmNleftWidget, name1);
    XtSetArg(ags[4], XmNtopAttachment, XmATTACH_FORM);
    XtSetArg(ags[5], XmNtopOffset, 5);
    XtSetArg(ags[6], XmNrightAttachment, XmATTACH_FORM);
    XtSetArg(ags[7], XmNvalue, point->author);
    name2 = XmCreateText(form1, "text", ags, 8);

    /* When the enter key is pressed the focus is moved to next
     * text widget. This functionality is set now.
     */

    XtAddCallback(name2, XmNactivateCallback, XmProcessTraversal,
        XmTRAVERSE_NEXT_TAB_GROUP);
    XtFree(one);
    one = XmStringCreateSimple("EMAIL ADDRESS");

    /* Create the e-mail label widget. */

    addr1 = XtVaCreateWidget("label", xmLabelWidgetClass, form1,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNtopAttachment, XmATTACH_WIDGET,
        XmNtopWidget, name1, XmNtopOffset, 25, XmNheight,
        25, NULL);

    /* Create the e-mail text widget. */

    XtSetArg(ags[0], XmNrows, 1);
    XtSetArg(ags[1], XmNcolumns, 20);
    XtSetArg(ags[2], XmNleftAttachment, XmATTACH_WIDGET);
    XtSetArg(ags[3], XmNleftWidget, addr1);
    XtSetArg(ags[4], XmNtopAttachment, XmATTACH_WIDGET);
    XtSetArg(ags[5], XmNtopWidget, name2);
    XtSetArg(ags[6], XmNtopOffset, 10);
    XtSetArg(ags[7], XmNrightAttachment, XmATTACH_FORM);
    XtSetArg(ags[8], XmNvalue, point->email);
    addr2 = XmCreateText(form1, "text", ags, 9);
    XtAddCallback(addr2, XmNactivateCallback, XmProcessTraversal,

```



```

        XmTRAVERSE_NEXT_TAB_GROUP);
XtFree(one);

/* Create the function label and text widget. */

one = XmStringCreateSimple("FUNCTION");
func1 = XtVaCreateWidget("label", xmLabelWidgetClass, form1,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNtopAttachment, XmATTACH_WIDGET,
        XmNtopWidget, addr1, XmNtopOffset, 25, XmNheight,
        15, NULL);
XtSetArg(ags[0], XmNrows, 1);
XtSetArg(ags[1], XmNcolumns, 20);
XtSetArg(ags[2], XmNleftAttachment, XmATTACH_WIDGET);
XtSetArg(ags[3], XmNleftWidget, func1);
XtSetArg(ags[4], XmNtopAttachment, XmATTACH_WIDGET);
XtSetArg(ags[5], XmNtopWidget, addr2);
XtSetArg(ags[6], XmNtopOffset, 10);
XtSetArg(ags[7], XmNrightAttachment, XmATTACH_FORM);
XtSetArg(ags[8], XmNvalue, point->function);
func2 = XmCreateText(form1, "text", ags, 9);
XtAddCallback(func2, XmNactivateCallback, XmProcessTraversal,
        XmTRAVERSE_NEXT_TAB_GROUP);
XtFree(one);

/* Create the method label and text widget. */

one = XmStringCreateSimple("METHOD");
method1 = XtVaCreateWidget("label", xmLabelWidgetClass,
        form1, XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNtopAttachment, XmATTACH_WIDGET,
        XmNtopWidget, func1, XmNtopOffset, 15, XmNheight,
        25, NULL);
XtSetArg(ags[0], XmNrows, 1);
XtSetArg(ags[1], XmNcolumns, 20);
XtSetArg(ags[2], XmNleftAttachment, XmATTACH_WIDGET);
XtSetArg(ags[3], XmNleftWidget, method1);
XtSetArg(ags[4], XmNtopAttachment, XmATTACH_WIDGET);
XtSetArg(ags[5], XmNtopWidget, func2);
XtSetArg(ags[6], XmNtopOffset, 10);
XtSetArg(ags[7], XmNrightAttachment, XmATTACH_FORM);
XtSetArg(ags[8], XmNvalue, point->method);
method2 = XmCreateText(form1, "text", ags, 9);
XtAddCallback(method2, XmNactivateCallback, XmProcessTraversal,
        XmTRAVERSE_NEXT_TAB_GROUP);
XtFree(one);

/* Create the implementation details label and text widget. */

one = XmStringCreateSimple("IMPLEMENTATION DETAILS:");
labell = XtVaCreateWidget("label", xmLabelWidgetClass, rowcol2,
        XmNlabelString, one, NULL);
XtSetArg(ags[0], XmNrows, 5);
XtSetArg(ags[1], XmNcolumns, 10);
XtSetArg(ags[2], XmNeditable, True);
XtSetArg(ags[3], XmNeditMode, XmMULTI_LINE_EDIT);
XtSetArg(ags[4], XmNwordWrap, True);
XtSetArg(ags[5], XmNvalue, point->implementation);
text_w1 = XmCreateScrolledText(rowcol2, "text", ags, 6);
rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcol2,
        NULL);

/* Create the OK and cancel buttons. */

one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass,
        rowcol3, XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNheight, 25, NULL);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass,
        rowcol3, XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
        XmNrightOffset, 5, XmNheight, 25, NULL);

XtAddCallback(pb1, XmNactivateCallback, change_desc, dialog1);
XtAddCallback(pb2, XmNactivateCallback, destroy, dialog1);

```

```

        XtFree(one);
        XtFree(two);
        XtManageChild(text_w1);
        XtManageChild(label1);
        XtManageChild(name1);
        XtManageChild(name2);
        XtManageChild(addr1);
        XtManageChild(addr2);
        XtManageChild(func1);
        XtManageChild(func2);
        XtManageChild(method1);
        XtManageChild(method2);
        XtManageChild(form1);
        XtManageChild(rowcol2);
        XtManageChild(rowcol3);
        XtManageChild(dialog1);
    } else
        errorDialog("FILE NOT FOUND ...");
}

/* This function is called once the user changes the description and pushes the
 * OK button in the above dialog . The length of each part of the description is
 * checked before changes are made to the file information. Only the library
 * administrator can change the description. For all other users, a note is made
 * in the update.dat file for the library administrator.
 */

int change_desc(Widget w_id, XtPointer w , XmAnyCallbackStruct *cbs)
{
    char *text, *text1, *text2, *text3, *text4;
    char str[100];
    FILE *fp;

    /* Get the strings from the text widgets. */

    text = XmTextGetString(name2);
    text1 = XmTextGetString(addr2);
    text2 = XmTextGetString(func2);
    text3 = XmTextGetString(method2);
    text4 = XmTextGetString(text_w1);

    /* Check the lengths of the strings. */

    if (strlen(text) < 1 || strlen(text1) < 1 || strlen(text2) < 1) {
        errorDialog("name,e-mail address, and function description must be
        provided");
        XtManageChild(w);
        return;
    }
    if (strlen(text) > 19) {
        errorDialog("NAME IS TOO LONG ....");
        return;
    }
    if (strlen(text1) > 40) {
        errorDialog("E_MAIL ADDRESS IS TOO LONG.....");
        return;
    }
    if (strlen(text2) > 99) {
        errorDialog("FUNCTION IS TOO LONG ....");
        return;
    }
    if (strlen(text3) > 99) {
        errorDialog("METHOD IS TOO LONG ....");
        return;
    }
    if (strlen(text4) > 999) {
        errorDialog("IMPLEMENTATION DETAILS IS TOO LONG .....");
        return;
    }
    if (sys_admin) {
        /* The user is the library administrator, so change the file
         * information.
         */
        point = NULL;
        traverse(head->left, file);
    }
}

```

```

    if (point) {
        strcpy(point->author, text);
        strcpy(point->email, text1);
        strcpy(point->function, text2);
        strcpy(point->method, text3);
        strcpy(point->implementation, text4);
    }
} else {

/* The user is not the library administrator, so make a note of the
 * required changes in the update.dat file in the library directory.
 */

sprintf(str, "%supdate.dat", lib_d);
fp = fopen(str, "r+");
if (!fp) {
    errordialog("UNABLE TO OPEN update file ...change not recorded");
} else {
    if (lock(fp)) {
        fseek(fp, 0, 2);
        sprintf(str, "Change description of file %s as follows\n",
            file);
        fputs(str, fp);
        sprintf(str, "name: %s\n", text);
        fputs(str, fp);
        sprintf(str, "email: %s\n", text1);
        fputs(str, fp);
        sprintf(str, "function: %s\n", text2);
        fputs(str, fp);
        sprintf(str, "method: %s\n", text3);
        fputs(str, fp);
        sprintf(str, "implementation: %s\n", text4);
        fputs(str, fp);
        unlock(fp);
        fclose(fp);
    } else {
        errordialog("UNABLE TO LOCK update file ...OPERATION NOT
            SUCCESSFUL");
    }
}

XtFree(text);
XtFree(text1);
XtFree(text2);
XtFree(text3);
XtFree(text4);
XtUnmanageChild(w);
}

/* This function is called when the user selects "add to thesaurus" in the options dialog
 * and pushes the OK button. This function pops up a dilog to receive the word to be added
 * to the thesaurus.
 */

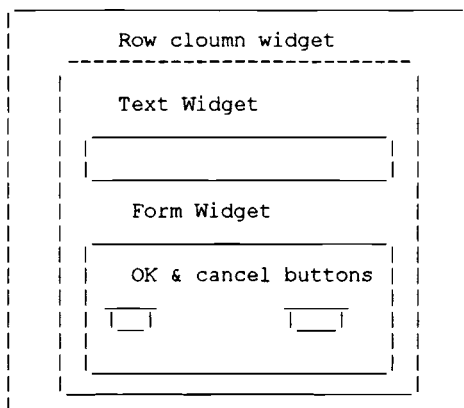
```

```

/*
The structure of this widget is:

```

Bulletin Board Widget



```

*/

void get_word()
{
    Widget      top, pb1, pb2, rowcol3, labell1, l1, p1;
    Arg         args[10];
    int         i = 0;
    XmString    one, two;

    XtSetArg(args[0], XmNautoUnmanage, True);
    XtSetArg(args[1], XmNdefaultPosition, False);
    XtSetArg(args[2], XmNallowShellResize, True);
    XtSetArg(args[3], XmNwidth, 400);
    XtSetArg(args[4], XmNheight, 300);
    XtSetArg(args[5], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);

    top = XmCreateBulletinBoardDialog(toplevel, "LIST", args, 5);

    /* Position the dialog at 250,400. */

    XtAddCallback(top, XmNmapCallback, map, 250400);
    XtAddCallback(top, XmNhelpCallback, help, 20);

    l1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, top, XmNnumColumns,
        1, XmNorientation, XmVERTICAL, NULL);

    one = XmStringCreateSimple("Give the word to be included:");
    labell1 = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
        one, NULL);

    XtFree(one);
    p1 = XtVaCreateWidget("text", xmTextWidgetClass, l1, NULL);
    XtAddCallback(p1, XmNactivateCallback, eq_word, p1);

    rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, l1, NULL);
    one = XmStringCreateSimple("OK");
    two = XmStringCreateSimple("CANCEL");

    pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNheight, 25, NULL);

    pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
        XmNrightOffset, 5, XmNheight, 25, NULL);

    XtAddCallback(pb1, XmNactivateCallback, eq_word, p1);
    XtAddCallback(pb2, XmNactivateCallback, destroy, top);

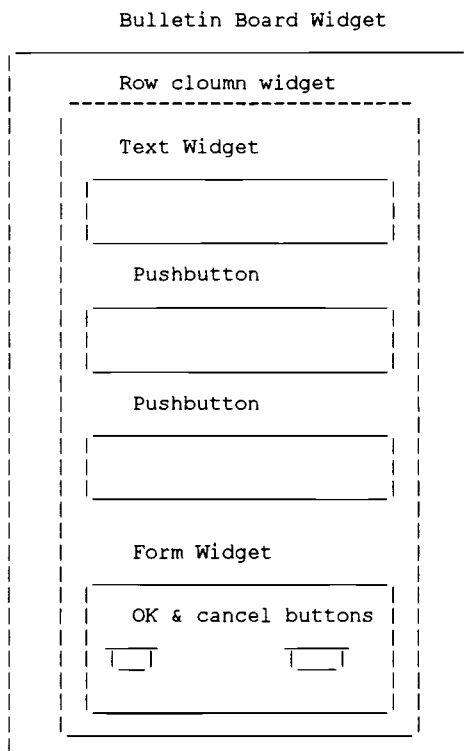
    XtManageChild(rowcol3);
    XtManageChild(labell1);
    XtManageChild(p1);
    XtManageChild(l1);
    XtManageChild(top);
}

/*
 * This function is called after the word to be added to the thesaurus is given
 * by the user. This function pops up a dialog with a text widget, two
 * pushbuttons, and a list widget. When the user pushes the add pushbutton,
 * the word in the text widget is added to the list. When the delete
 * pushbutton is pushed, the word in the text widget is deleted from the
 * list. When the OK Pushbutton is pushed, the words in the list are recorded
 * as equivalent words.
 */

```

/*

The structure of this dialog box is as follows.



*/

```

void eq_word(Widget tw, Widget w, XmAnyCallbackStruct *cbs)
{
    --
    char *text;
    Arg args[10];
    int i = 0;
    XmString one, two;

    text = XmTextGetString(w);
    strcpy(th word, text);
    XtFree(text);

    XtSetArg(args[0], XmNautoUnmanage, True);
    XtSetArg(args[1], XmNdefaultPosition, False);
    XtSetArg(args[2], XmNallowShellResize, True);
    XtSetArg(args[3], XmNwidth, 400);
    XtSetArg(args[4], XmNheight, 300);

    top = XmCreateBulletinBoardDialog(toplevel, "LIST", args, 5);
    /* Position the dialog at 250,100. */
    XtAddCallback(top, XmNmapCallback, map, 250100);
    XtAddCallback(top, XmNhelpCallback, help, 21);
    l1 = XtVaCreateWidget("rowcol", xmFormWidgetClass, top, NULL);
    /* Create the "equivalent word" label widget. */
    one = XmStringCreateSimple("Equivalent word:");
    labell = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
  
```

```

        one, XmNleftAttachment, XmATTACH_FORM, XmNleftOffset, 10, XmNtopAttachment,
        XmATTACH_FORM, XmNtopOffset, 10, XmNheight, 25, NULL);

/* Create the text widget. */

p1 = XtVaCreateManagedWidget("text", xmTextWidgetClass, l1, XmNleftAttachment,
        XmATTACH_WIDGET, XmNleftWidget, label1, XmNleftOffset, 10, XmNtopAttachment,
        XmATTACH_FORM, XmNtopOffset, 5, XmNrightAttachment, XmATTACH_FORM,
        XmNrightOffset, 10, NULL);

/* Create the "ADD" pushbutton. */

one = XmStringCreateSimple("    ADD    ");
pb3 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, l1,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM, XmNleftOffset,
        10, XmNtopAttachment, XmATTACH_WIDGET, XmNtopWidget, label1, XmNtopOffset,
        15, XmNrightAttachment, XmATTACH_FORM, XmNrightOffset, 10, NULL);
XtAddCallback(pb3, XmNactivateCallback, add_word, top);

/* Create the delete pushbutton. */

one = XmStringCreateSimple("    delete    ");
pb4 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, l1,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM, XmNleftOffset,
        10, XmNtopAttachment, XmATTACH_WIDGET, XmNtopWidget, pb3, XmNtopOffset,
        15, XmNrightAttachment, XmATTACH_FORM, XmNrightOffset, 10, NULL);
XtAddCallback(pb4, XmNactivateCallback, delete_word, p1);
XtSetArg(args[0], XmNleftAttachment, XmATTACH_FORM);
XtSetArg(args[1], XmNleftOffset, 10);
XtSetArg(args[2], XmNrightAttachment, XmATTACH_FORM);
XtSetArg(args[3], XmNrightOffset, 10);
XtSetArg(args[4], XmNtopAttachment, XmATTACH_WIDGET);
XtSetArg(args[5], XmNtopWidget, pb4);
XtSetArg(args[6], XmNtopOffset, 15);

/* Create the list widget. */

list_w = XmCreateScrolledList(l1, "list_w", args, 7);
XtVaSetValues(list_w, XmNvisibleItemCount, 10, NULL);
rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, l1, XmNtopAttachment,
        XmATTACH_WIDGET, XmNtopWidget, list_w, XmNtopOffset, 25, XmNleftAttachment,
        XmATTACH_FORM, XmNrightAttachment, XmATTACH_FORM, NULL);
one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNheight, 25, NULL);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
        XmNrightOffset, 5, XmNheight, 25, NULL);
XtAddCallback(pb1, XmNactivateCallback, update_thesaurus, top);
XtAddCallback(pb2, XmNactivateCallback, destroy, top);
XtManageChild(rowcol3);
XtManageChild(list_w);
XtManageChild(label1);
XtManageChild(l1);
XtManageChild(top);
}

/* This function is called when the user pushes the Add pushbutton.
 * The word in the text widget is added to the list widget.
 */

void add_word(Widget tw, XtPointer unused, XmPushButtonCallbackStruct *cbs)
{
    char    *text;
    XmString word;

    /* Get the word from the text widget. */

    text = XmTextGetString(p1);
    word = XmStringCreateSimple(text);

    /* Add the word to the list widget. */

    XmListAddItemUnselected(list_w, word, 1);
    XmTextSetString(p1, "");
}

```

```

}

/* This function is called when the delete pushbutton is pushed.
 * The word in the text widget is deleted from the list.
 */

void delete_word(Widget tw, XtPointer unused, XmPushButtonCallbackStruct *cbs)
{
    char *text;
    XmString word;

    /* Get the word from the text widget. */

    text = XmTextGetString(pl);
    word = XmStringCreateSimple(text);

    /* Delete the word from the list */

    XmListDeleteItem(list_w, word, 1);
    XmTextSetString(pl, "");
}

/* This function is called when the user pushes the OK button in the equivalent
 * words dialog. The words in the list widget are recorded as equivalent words
 * to the given word in the thesaurus.
 * Note that any changes to words already present needs the approval of the
 * library administrator.
 */

void update_thesaurus(Widget tw, Widget unused, XmPushButtonCallbackStruct *cbs)
{
    FILE *fp;
    char str[100], *text;
    XmStringTable choice;
    int count, flag, i = 0;
    XtVaGetValues(list_w, XmNitemCount, &count, XmNitems, &choice, NULL);
    flag = 0;
    XtUnmanageChild(unused);

    /* If the word is already present make a note in the update.dat else
     * change the thesaurus.dat file.
     */

    if (!thesaurus(th_word))
        sprintf(str, "%sthesaurus.dat", lib_d);
    else {
        errorDialog("word already present update will not be visible
        immediately");
        sprintf(str, "%supdate.dat", lib_d);
        flag = 1;
    }
    fp = fopen(str, "r+");
    if (!fp) {
        errorDialog("error opening thesaurus file");
    } else {

        /* Append at the end of the file. */

        if (lock(fp)) {
            fseek(fp, 0, 2);
            if (flag)
                fputs("update Thesaurus word ", fp);
            fputs(th_word, fp);
            fputs("\n", fp);
            for (i = 0; i < count; i++) {
                XmStringGetLtoR(choice[i], charset, &text);
                fputs(text, fp);
                XtFree(text);
                fputs(" ", fp);
            }
            sprintf(str, ".\n");
            fputs(str, fp);
            unlock(fp);
            fclose(fp);
        } else
            errorDialog("unable to lock thesaurus file..");
    }
}

```

```

}

/*
* * * * *
*   Filename : Stat.c
*
*   Programmed by : N. Sunil Chakravarthy
*
*   Last updated on : 08.17.94
* * * * *
*/

```

This file contains the code to display the different graphs available to be viewed.
The file also contains the necessary functions to display the graphs.

```

#include <Xm/DialogS.h>
#include <Xm/TextF.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/Form.h>
#include <Xm/SelectioB.h>
#include <Xm/PushB.h>
#include <Xm/LabelG.h>
#include <stdio.h>
# include <math.h>

void show_graph(), space_graph();
void frequency();
void plot_freq();
void setxy();
void nofiles();
void names();
void display_list();
int  coutfreq();
int  freq_graph();
extern rowcol;
extern float *setx, *sety;
extern int  tcount;

extern void destroy();
extern void Destroy();
extern void help();
extern void toggle();
extern space_save_graph();
extern add_list();

extern lock();
extern unlock();

extern struct file_node {
    char  name[20];
    char  rcsfile[80];
    char  rcsno[2000];
    char  author[20];
    char  email[20];
    char  function[100];
    char  method[100];
    char  implementation[1000];
    float saved;
    int   outno;
    int   status;
    int   filesize;
    struct file_node *left;
    struct file_node *right;
    struct file_node *parent;
};

extern struct list {
    char  name[20];
    char  filename[20];
    int   type;
    int   distance;
    int   filesize;
    struct list *left;
    struct list *right;
};

```



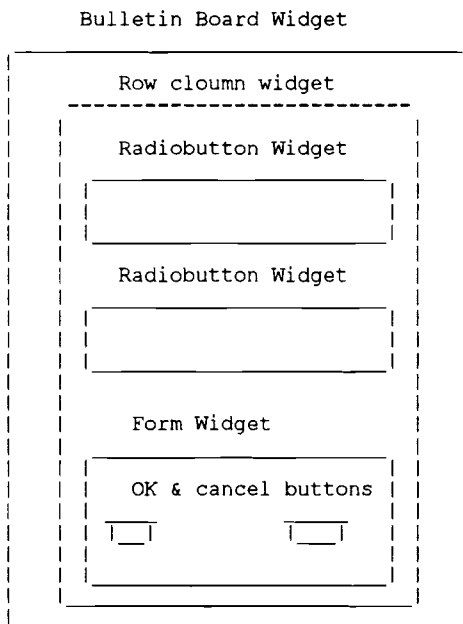
```
extern struct list *listhead;

extern struct file_node *new_file, *point, *head, *disp_node;
extern char lib_d[100];
extern map();

extern Widget toplevel;
extern int mode;
int usertime = 0;
int cind = 0;

/* This function is called when the stat drawn button is pushed.
 * This function displays a dialog with two radiobuttons representing
 * two choices: "space saved graph" and "frequency of checkout graph".
 */

/*
  The structure of this widget is:
```



```
*/

void stat_button(Widget w)
{
  Widget dialog, pb1, cin_R, cin_N, cin_S, cin_T, cin_M, rowcoll, pb2,
        rowcol3, rad;
  XmString one, two, three, four, five;
  Arg args[10];
  int i;
  i = 0;
  XtSetArg(args[i], XmNautoUnmanage, True);
  i++;
  XtSetArg(args[i], XmNdefaultPosition, False);
  i++;
  XtSetArg(args[i], XmNallowShellResize, True);
  i++;
  XtSetArg(args[i], XmNwidth, 100);
  i++;
  XtSetArg(args[i], XmNheight, 100);
  i++;
  XtSetArg(args[i], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);

  dialog = XmCreateBulletinBoardDialog(w, "STATISTICS", args, 6);

  /* Position the dialog at 150,500. */

  XtAddCallback(dialog, XmNmapCallback, map, 150500);
}
```

```

XtAddCallback(dialog, XmNhelpCallback, help, 16);

rowcol1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, dialog,
                          XmNnumColumns, 1, XmNpacking, XmPACK_COLUMN, XmNorientation,
                          XmVERTICAL, NULL);
one = XmStringCreateSimple("SPACE SAVED GRAPH");
two = XmStringCreateSimple("CHECKOUT FREQUENCY GRAPH");
three = XmStringCreateSimple("SPACE SAVED IN BYTES");
mode = 0;
rad = XmVaCreateSimpleRadioBox(rowcol1, "radio_box", 0, toggle, XmVaRADIOBUTTON,
                              one, 0, NULL, NULL, XmVaRADIOBUTTON, two, 0, NULL, NULL, NULL);

rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcol1,
                          NULL);

one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
                              XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
                              XmNleftOffset, 5, XmNheight, 25, XmNwidth, 50);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
                              XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
                              XmNrightOffset, 5, XmNheight, 25, XmNwidth, 50);

XtAddCallback(pb1, XmNactivateCallback, show_graph, dialog);
XtAddCallback(pb2, XmNactivateCallback, destFoy, dialog);
XmStringFree(one);
XtManageChild(rowcol1);
XtManageChild(rowcol3);
XtManageChild(rad);
XtManageChild(dialog);
}

/* This function is called when the frequency of checkout graph radiobutton
 * is pushed. In this function, mode is set to 1, which is used later when the OK
 * button is clicked.
 */

void space_graph(Widget w_id, XtPointer client_data, XmPushButtonCallbackStruct *cbs)
{
    mode = 1;
}

/* This function is called when the OK button is pushed in the stat dialog. */

void show_graph(Widget w_id, XtPointer client_data, XmPushButtonCallbackStruct *cbs)
{
    Widget wid = client_data;
    if (mode == 0) {

        /* The user wants to see the space utilization graph. */

        XtUnmanageChild(client_data);

        /* Display the list of files. */

        display_list();
        XSync(XTDisplay(w_id), False);

        /* Display the graph. */

        space_save_graph();

    }

    /* If mode=1, the user wants to see the frequency of
     * checkout graph.
     */

    if (mode == 1)
        frequency(wid);
}

/* This function is called to receive the number of days from which
 * to plot the frequency of check out. Note that part of a day can
 * also be given. For example, giving 3.5 means plot the frequency

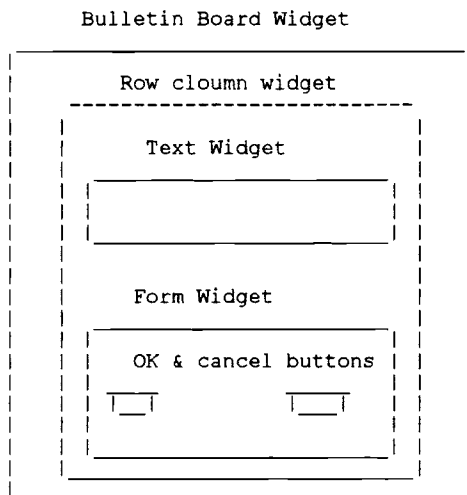
```

* of check out for the last 3.5 days.

*/

/*

The structure of this widget is:



*/

```

void frequency(Widget wid)
{
    Widget      top, pb1, pb2, rowcol3, labell, l1, p1;
    Arg  args[10];
    int   i = 0;
    XmString one, two;
    XtSetArg(args[0], XmNautoUnmanage, True);
    XtSetArg(args[1], XmNdefaultPosition, False);
    XtSetArg(args[2], XmNallowShellResize, True);
    XtSetArg(args[3], XmNwidth, 400);
    XtSetArg(args[4], XmNheight, 300);
    XtSetArg(args[5], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);

    top = XmCreateBulletinBoardDialog(XtParent(wid), "LIST", args, 5);
    XtAddCallback(top, XmNmapCallback, map, 250400);
    XtAddCallback(top, XmNhelpCallback, help, 22);

    XtUnmanageChild(wid);

    l1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, top, XmNnumColumns,
        1, XmNorientation, XmVERTICAL, NULL);

    one = XmStringCreateSimple("Give the number of days");
    labell = XtVaCreateWidget("label", xmLabelWidgetClass, l1, XmNlabelString,
        one, NULL);

    XtFree(one);
    p1 = XtVaCreateWidget("text", xmTextWidgetClass, l1, NULL);

    rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, l1, NULL);
    one = XmStringCreateSimple("OK");
    two = XmStringCreateSimple("CANCEL");
    pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNheight, 25, NULL);

    pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
        XmNrightOffset, 5, XmNheight, 25, NULL);
    XtAddCallback(pb1, XmNactivateCallback, plot_freq, p1);
    XtAddCallback(pb2, XmNactivateCallback, destFoy, top);
    XtManageChild(rowcol3);
    XtManageChild(labell);
    XtManageChild(p1);
    XtManageChild(l1);
    XtManageChild(top);
}

```

```

}

/* This function is called when the OK button is pushed in the above (no of days)
 * dialog. A conversion is done to time in seconds from Jan 1st 1970. This is
 * done because each check-out is recorded using the time function.
 */

void plot_freq(Widget tw, XtPointer w, XmTextVerifyCallbackStruct *cbs)
{
    char    *text;
    time_t  *tloc, t;
    int     secs, newsec = 0;
    double  days = 0.0;

    text = XmTextGetString(w);
    XtUnmanageChild(XtParent(XtParent(tw)));
    days = atof(text);
    secs = (int)(days * 86400);
    tloc = (time_t *)malloc(sizeof(time_t));
    if (tloc) {
        t = time(tloc);
        newsec = t - secs;

        /* Calculate the time in secs of the number given. */

        if (newsec > 0) {

            /* If this number is positive then the user has
             * entered valid number in no of days dialog.
             */

            usertime = newsec;
            display_list();
            XSync(XtDisplay(w), False);
            freq_graph();
        }
    }
}

/* This function is used to generate the necessary information in a file to
 * plot the graph using BLT. The file, when executed at the shell, creates the
 * graph.
 */

int freq_graph()
{
    FILE * ofp;
    char  line1[200], temp[80];
    int   j = 0;

    void nofiles();
    void setxy();
    sprintf(temp, "%sfreq.grph", lib_d);
    ofp = fopen(temp, "w");
    if (!ofp) {
        errorDialog("Unable to open plot file ..");
        return;
    }

    /* Lock the freq.grph file in the library directory to avoid
     * simultaneous updates.
     */

    if (!lock(ofp)) {
        errorDialog("Unable to lock plot file ..");
        return;
    }

    fprintf(ofp, "#!/contrib/bin/blt_wish -f\n\n");
    fprintf(ofp, "if [file exists /contrib/library] {\n");
    fprintf(ofp, "    set blt_library /contrib/library\n\n");
    fprintf(ofp, "option add *Blt_hText.Font *Times-Bold-R*14*\n");
    fprintf(ofp, "option add *Blt_text.Font *Times-Bold-R*12*\n");
    fprintf(ofp, "option add *graph.xTitle \"File #\"\n");
    fprintf(ofp, "option add *graph.yTitle \"No of Times Checked Out\"\n");
    fprintf(ofp, "option add *graph.title \"Figure 2. Checkout Frequency\"\n");
}

```

```

fprintf(ofp, "option add *Blt_graph.legendFont *Times--*-8*\n\n");
fprintf(ofp, "set visual [wininfo screenvisual .]\n");
fprintf(ofp, "if { $visual != \"staticgray\" } { \n");
fprintf(ofp, "    option add *print.background yellow \n");
fprintf(ofp, "    option add *quit.background red \n");
fprintf(ofp, "}\n\n");
fprintf(ofp, "global graph\n");
fprintf(ofp, "set graph .graph\n");
fprintf(ofp, "blt_htext .header -text {#####\n");
strcpy(line1, "Plot For VCI Check out Frequency\n");
fprintf(ofp, "##### %s", line1);
fprintf(ofp, "To create a postscript file \"freq.ps.\", press the #####\n");
fprintf(ofp, "button $blt_htext(widget).print -text print -command {\n");
fprintf(ofp, "    .graph postscript freq.ps -pagewidth 6i -pageheight 4i");
fprintf(ofp, " -landscape false \n }\n\n");
fprintf(ofp, "$blt_htext(widget) append $blt_htext(widget).print\n");
fprintf(ofp, "##### button.\n\n");
fprintf(ofp, "blt_graph $graph\n\n");
fprintf(ofp, "blt_htext .footer -text {Hit the #####\n");
fprintf(ofp, "button $blt_htext(widget).quit -text quit -command {destroy .}\n");
fprintf(ofp, "$blt_htext(widget) append $blt_htext(widget).quit\n");
fprintf(ofp, "##### button when you are done.#####\n");
fprintf(ofp, "$blt_htext(widget) -padx 20\n");
fprintf(ofp, "#####}\n\n");

/* Count the number of files in the library. */

tcount = 0;
nofiles(head->left);

/* Create two dynamic arrays for x and y values. */

setx = (float *) malloc(sizeof(float)*tcount);
sety = (float *) malloc(sizeof(float)*tcount);
cind = 0;

/* Load x and y values. */

setxy(head->left);

/* Print x and y values in the file. */

fprintf(ofp, "set X1 { \n %2.1f %2.1f", setx[0], setx[0]);
for (j = 1; j < tcount; j++)
    fprintf(ofp, " %2.1f", setx[j]);
fprintf(ofp, "\n\n\n set F1 {\n %f\n %f\n", sety[0], sety[0]);
for (j = 1; j < tcount; j++)
    fprintf(ofp, " %f\n", sety[j]);
fprintf(ofp, "\n\n\n");

fprintf(ofp, "$graph element create {} -xdata $X1 -ydata $F1 \\ \n");
fprintf(ofp, "    -symbol circle -linewidth 1\n");
fprintf(ofp, "# $graph crosshairs set on\n\n");

fprintf(ofp, "pack append . \\ \n .header { padx 20 pady 10 } \\ \n");
fprintf(ofp, "    .graph { fill expand } \\ \n .footer { padx 20 pady 10 }\n\n ");

fprintf(ofp, "wm min . 0 0\n\n");

fprintf(ofp, "bind $graph <B1-ButtonRelease> { ##W crosshairs toggle }\n\n");

fprintf(ofp, "proc TurnOnHairs { graph } {\n");
fprintf(ofp, "    bind $graph <Any-Motion> {##W crosshairs configure -position @%
    %x,%y}\n }\n\n");
fprintf(ofp, "proc TurnOffHairs { graph } {\n");
fprintf(ofp, "    bind $graph <Any-Motion> {##W crosshairs configure -position @%
    %x,%y} \n }\n\n");
fprintf(ofp, "bind $graph <Enter> { TurnOnHairs ##W }\n");
fprintf(ofp, "bind $graph <Leave> { TurnOffHairs ##W }\n\n");
unlock(ofp);
fclose(ofp);
memset(temp, '\0', 80);
sprintf(temp, "chmod 777 %sfreq.grph", lib_d);
system(temp);
memset(temp, '\0', 80);

/* Execute the freq.grph graph at the shell to show the graph. */

```

```

        sprintf(temp, "%sfreq.grph &", lib_d);
        system(temp);
    }

/* This function is used to set x and y values for the frequency of check-out
 * graph.
 */

void setxy(struct file_node *cur)
{
    if (cur) {
        setx[cind] = cind;

        /* The number of times this (cur) file was checked out is found
         * using the coutfreq function.
         */

        sety[cind++] = coutfreq(cur->name);
        setxy(cur->left);
        setxy(cur->right);
    }
}

/* This function is used to count the number of files in the library.
 * This a recursive function and sets tcount to the number.
 */

void nofiles(struct file_node *cur)
{
    if (cur) {
        tcount++;
        countfiles(cur->left);
        countfiles(cur->right);
    }
}

/* This function is used to calculate the number of times
 * a given file is checked-out in a given period of time.
 * The checkout log file is searched and for every occurrence
 * of this file, the time of check-out is checked.
 * If check-out time is within the stipulated period indicated
 * by usertime, the count is incremented.
 */

int    coutfreq(char name[100])
{
    FILE * fp;
    char  str[100], fname[100];
    int   ftime, ccount = 0;

    sprintf(str, "%scout.dat", lib_d);
    fp = fopen(str, "r");
    if (!fp) {
        printf("error opening cout.dat file\n");
        exit(0);
    }
    fgets(str, 80, fp);
    while (!feof(fp)) {
        sscanf(str, "%s %d", fname, &ftime);
        if (strcmp(fname, name) == 0) {

            /* One instance of check-out log of file
             * is found. Check the time.
             */

            if (ftime > usertime)
                ccount++;
        }
        memset(str, '\0', 100);
        fgets(str, 80, fp);
    }
    fclose(fp);
    return(ccount);
}

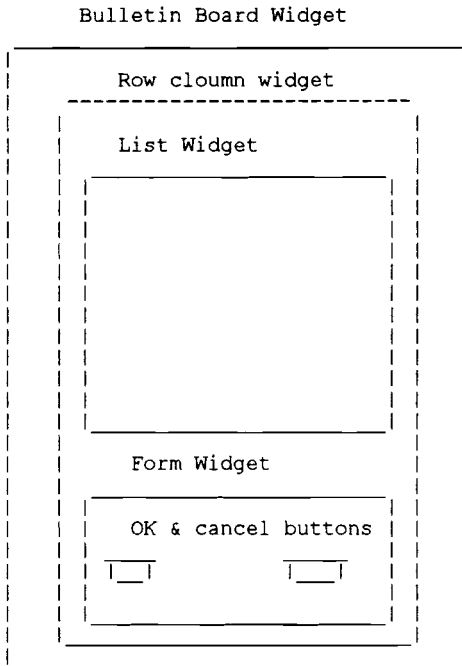
```

```

/* This function is used to display the list of files at the side of the
 * two graphs. In the graphs, the list have to be sorted by their file
 * size.
 */

```

The structure of this widget is:



```

*/

```

```

void display_list()
{
    Arg args[10];
    int i = 0;
    XmString one, two;
    Widget top, l1, list w, rowcol3, pb1, pb2, labell1;
    XtSetArg(args[i], XmNautoUnmanage, True);
    i++;
    XtSetArg(args[i], XmNdefaultPosition, False);
    i++;
    XtSetArg(args[i], XmNallowShellResize, True);
    i++;
    XtSetArg(args[i], XmNwidth, 400);
    i++;
    XtSetArg(args[i], XmNheight, 300);
    i++;

    top = XmCreateBulletinBoardDialog(toplevel, "LIST_OF_FILES", args, 5);

    /* Position the dialog at 450,100. */

    XtAddCallback(top, XmNmapCallback, map, 450100);
    XtAddCallback(top, XmNhelpCallback, help, 3);

    l1 = XtVaCreateWidget("rowcol", xmFormWidgetClass, top, NULL);

    XtSetArg(args[0], XmNleftAttachment, XmATTACH_FORM);
    XtSetArg(args[1], XmNleftOffset, 10);
    XtSetArg(args[2], XmNrightAttachment, XmATTACH_FORM);
    XtSetArg(args[3], XmNrightOffset, 10);
    XtSetArg(args[4], XmNtopAttachment, XmATTACH_FORM);
    XtSetArg(args[5], XmNtopOffset, 15);

    list w = XmCreateScrolledList(l1, "list_w", args, 6);
    XtVaSetValues(list_w, XmNvisibleItemCount, 10, NULL);
}

```

```

rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, ll, XmNtopAttachment,
    XmATTACH_WIDGET, XmNtopWidget, list_w, XmNtopOffset, 25, XmNleftAttachment,
    XmATTACH_FORM, XmNrightAttachment, XmATTACH_FORM, NULL);
one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateWidget("push", xmPushButtonWidgetClass, rowcol3,
    XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM, XmNleftOffset,
    5, XmNheight, 25, NULL);

pb2 = XtVaCreateWidget("push", xmPushButtonWidgetClass, rowcol3,
    XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM, XmNrightOffset,
    5, XmNheight, 25, NULL);
XtAddCallback(pb1, XmNactivateCallback, destroy, top);
XtAddCallback(pb2, XmNactivateCallback, destroy, top);
XtManageChild(pb1);
XtManageChild(pb2);
XtManageChild(rowcol3);
XtManageChild(list_w);
XtManageChild(ll);
XtManageChild(top);

/* The following function is used to load the names of the
 * files in the required order.
 */

names(list_w);
XSync(XtDisplay(top), False);
}

/* This function is used to sort the files by their size. */
void names(Widget list_w)
{
    struct list *p, *q, *r, *s, *qprev, *rprev, *temp, *t;
    int    flag = 0;
    int    count = 0;
    char   str[100];
    XmString listname;
    int    i;

    listhead = (struct list *) malloc(sizeof(struct list ));
    if (!listhead) {
        errorDialog("UNABLE TO MALLOC");
    } else {
        p = listhead;
        add_list(p, head->left);

        qprev = listhead;

        /* Order the files by their sizes. */
        for (q = listhead->right; q; ) {
            qprev->right = q->right;
            t = q->right;
            if (qprev == listhead)
                listhead->right = q->right;
            r = listhead->right;
            rprev = listhead;
            while (r && (q->filesize > r->filesize)) {
                rprev = r;
                r = r->right;
            }
            s = rprev->right;
            rprev->right = q;
            if (rprev == listhead)
                listhead->right = q;
            q->right = s;
            q = t;
            temp = listhead;
            while (temp->right != t)
                temp = temp->right;
            qprev = temp;
        }

        i = 1;

```



```

/* Enter all the enteries from the above list into
 * the list widget after checking for duplicate enteries.
 */

p = listhead->right;
while (p != NULL) {
    q = listhead;
    for (; q == p; q = q->right) {
        if (strcmp(q->name, p->name) == 0)
            flag = 1;
    }
    if (!flag) {
        sprintf(str, "%d %s", count, p->name);

        /* show the filesize and filename */

        sprintf(str, "%d %s", p->filesize,
                p->name);
        listname = XmStringCreateSimple(str);
        count++;
        XmListAddItemUnselected(list_w, listname,
                                i++);
    }
    p = p->right;
}
}

/*
 * * * * *
 *   Filename : Help.c
 * * * * *
 *   Programmed by : N. Sunil Chakravarthy
 * * * * *
 *   Last updated on : 08.17.94
 * * * * *

```

This file has the code to pop up the dialog after the help drawn button is pushed.

```

*/

#include <Xm/DialogS.h>
#include <Xm/TextF.h>
#include <Xm/RowColumn.h>
#include <Xm/Text.h>
#include <Xm/Form.h>
#include <Xm/SelectioB.h>
#include <Xm/PushB.h>
#include <Xm/LabelG.h>
# include <stdio.h>
# include "readme.h"

void mode_set();
void get_file();
void change_description();
void get_status_file();
void change_status();

void help_ok();
void readme();

extern Widget rowcol;
extern Widget toplevel;

extern void destroy();
extern void Destroy();
extern void help();
extern void list_display();
extern void system_select();
extern void add_node();
extern struct file_node {

    char    name[20];
    char    rcsfile[80];
    char    rcsno[2000];

```

```

char    author[20];
char    email[20];
char    function[100];
char    method[100];
char    implementation[1000];
float   saved;
int     outno;
int     status;
int     filesize;

struct file_node *left;
struct file_node *right;
struct file_node *parent;
};

extern struct file_node *new_file, *point, *head;

extern char    dir[100], file_name[100], src_d[100], lib_d[100];
extern int     gl_test;
extern char    filename[80];

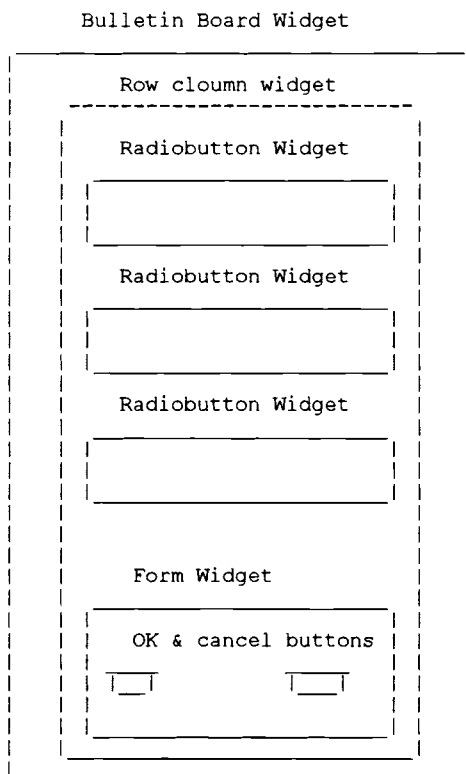
extern map();
extern toggle();

extern int     mode;

/* This function is called when the help drawn button is pushed.
 * This function displays three choices as radiobuttons in a dialog.
 * They are "Contents" , "Using help" and "About VCI".
 * The user is allowed to choose one of the above, and as soon
 * as the OK button is pushed, the text for the corresponding topics are
 * displayed in a dialog.
 */

```

The structure of this widget is as follows.



*/

```

Widget dialog;
void get_help()
{
    Widget pb1, cin_R, cin_N, cin_S, cin_T, cin_M, rowcol1, pb2, rowcol3,
        pb3, pb4, pb5, rad;
    XmString one, two, three, four, five;
    Arg args[10];
    int i;
    i = 0;

    XtSetArg(args[0], XmNautoUnmanage, True);
    XtSetArg(args[1], XmNdefaultPosition, False);
    XtSetArg(args[2], XmNallowShellResize, True);
    XtSetArg(args[3], XmNwidth, 200);
    XtSetArg(args[4], XmNheight, 100);
    XtSetArg(args[5], XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL);

    dialog = XmCreateBulletinBoardDialog(toplevel, "UTIL", args, 6);

    /* Position the dialog at 150,500. */

    XtAddCallback(dialog, XmNmapCallback, map, 150500);

    rowcol1 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, dialog,
        XmNnumColumns, 1, XmNpacking, XmPACK_COLUMN, XmNorientation,
        XmVERTICAL, NULL);
    one = XmStringCreateSimple("Contents");
    two = XmStringCreateSimple("Using Help");
    three = XmStringCreateSimple("About VCI");
    mode = 0;
    rad = XmVaCreateSimpleRadioBox(rowcol1, "radio_box", 0, toggle, XmVaRADIOBUTTON,
        one, 0, NULL, NULL, XmVaRADIOBUTTON, two, 0, NULL, NULL, XmVaRADIOBUTTON,
        three, 0, NULL, NULL, NULL);
    rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcol1, NULL);

    one = XmStringCreateSimple("OK");
    two = XmStringCreateSimple("CANCEL");
    pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNbottomAttachment, XmATTACH_FORM, XmNleftOffset, 5,
        XmNheight, 25, NULL);

    pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
        XmNbottomAttachment, XmATTACH_FORM, XmNrightOffset,
        5, XmNheight, 25, NULL);

    XtAddCallback(pb1, XmNactivateCallback, help_ok, dialog);
    XtAddCallback(pb2, XmNactivateCallback, destroy, dialog);
    XmStringFree(one);
    XmStringFree(two);
    XmStringFree(three);
    XtManageChild(rowcol1);
    XtManageChild(rowcol3);
    XtManageChild(rad);
    XtManageChild(dialog);
}

```

```

/* This function is called when the OK button is pushed in the above dialog.
 * The second and third choices are handled using regular help dialog
 * techniques. The "content" choice is handled in a separate dialog due to its
 * voluminous contents.
 */

```

```

void help_ok(Widget w_id, XtPointer client_data, XmPushButtonCallbackStruct*cbs)
{
    XtUnmanageChild(client_data);
    switch (mode) {

    case 0:
        readme();
        break;

```

```

case 1:
    help(toplevel, 17, NULL);
    break;

case 2:
    help(toplevel, 18, NULL);
    break;

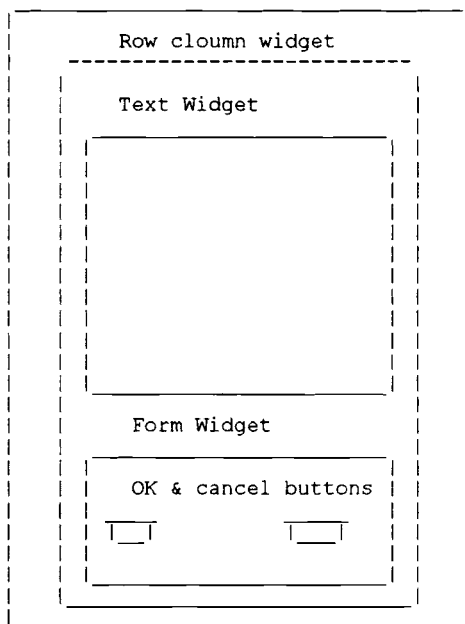
}

}

/* This function is called to display the contents (readme) of the help.*/
/*
The structure of this widget is as follows.

```

Bulletin Board Widget



```
*/
```

```

void readme()
{
    Widget dialog, text_w1, rowcol2, rowcol3, pb1, pb2;
    XmString text, one, two;
    Arg ags[10];
    char temp[15000];
    XtSetArg(ags[0], XmNautoUnmanage, True);
    XtSetArg(ags[1], XmNdefaultPosition, False);
    XtSetArg(ags[2], XmNallowShellResize, True);
    dialog = XmCreateBulletinBoardDialog(toplevel, "HELP", ags, 3);

    /* Position the dialog at 10,10. */

    XtAddCallback(dialog, XmNmapCallback, map, 010010);

    rowcol2 = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass, dialog,
        XmNnumColumns, 1, XmNorientation, XmVERTICAL, NULL);
    XtSetArg(ags[0], XmNrows, 30);
    XtSetArg(ags[1], XmNcolumns, 80);
    XtSetArg(ags[2], XmNeditable, False);
    XtSetArg(ags[3], XmNeditMode, XmMULTI_LINE_EDIT);
    XtSetArg(ags[4], XmNwordWrap, True);

    /* Get the string to be displayed. */

    sprintf(temp, "%s%s%s%s%s%s%s%s", readme_data[0], readme_data[1],
        readme_data[2], readme_data[3], readme_data[4], readme_data[5],

```

```

        readme_data[6], readme_data[7], readme_data[8], readme_data[9]);
XtSetArg(ags[5], XmNvalue, temp);
XtSetArg(ags[6], XmNscrollHorizontal, False);
text_w1 = XmCreateScrolledText(rowcol2, "text", ags, 7);
rowcol3 = XtVaCreateWidget("rowcol", xmFormWidgetClass, rowcol2,
        NULL);
one = XmStringCreateSimple("OK");
two = XmStringCreateSimple("CANCEL");
pb1 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, one, XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, XmNheight, 25, NULL);

pb2 = XtVaCreateManagedWidget("push", xmPushButtonWidgetClass, rowcol3,
        XmNlabelString, two, XmNrightAttachment, XmATTACH_FORM,
        XmNrightOffset, 5, XmNheight, 25, NULL);
XtAddCallback(pb1, XmNactivateCallback, destroy, dialog);
XtAddCallback(pb2, XmNactivateCallback, destroy, dialog);

XtFree(one);
XtFree(two);
XtManageChild(text_w1);
XtManageChild(rowcol2);
XtManageChild(rowcol3);
XtManageChild(dialog);
}

```

```
/*
```

```

* * * * *
*   Filename : Exit.c
*
*   Programmed by : N. Sunil Chakravarthy
*
*   Last updated on : 08.17.94
* * * * *

```

This file contains the code to pop up a question dialog, when the user pushes the Exit drawn button.

```
*/
```

```

#include <stdlib.h>
#include <Xm/DialogS.h>
extern Widget toplevel;

```

```

/* This function is called when the exit drawn button is pushed. A question
 * dialog is popped up, which asks the user if (s)he is sure that (s)he wants to
 * exit. If the user pushes the OK button, the application is closed.
 * If the user pushes the cancel Button, the user is returned to screen2.
 */

```

```
void exitdialog()
```

```

{
    Widget dialog;
    XmString text;
    dialog = XmCreateQuestionDialog(toplevel, "MESSAGE", NULL, 0);
    text = XmStringCreateSimple("ARE YOU SURE YOU WANT TO EXIT?");
    XtVaSetValues(dialog, XmNmessageString, text, NULL);
    XtUnmanageChild(XmMessageBoxGetChild(dialog, XmDIALOG_HELP_BUTTON));
    XtAddCallback(dialog, XmNokCallback, exit, NULL);
    XmStringFree(text);
    XtManageChild(dialog);
}

```

```
/*
```

```

* * * * *
*   Filename : Lock.c
*
*   Programmed by : N. Sunil Chakravarthy
*
*   Last updated on : 08.17.94
* * * * *

```

This file contains the code to Lock and Unlock a file.

```

*/

# include <fcntl.h>
# include <unistd.h>
# include <stdio.h>
# include <errno.h>

/* Max try is the maximum number of times the program tries to lock
 * a file before returning failure.
 */

#define MAX_TRY 10

extern errordialog();

/* This function is used to lock a file using the lockf function.
 * If the file is locked already by other user, sleep for 2
 * seconds and try again. The program tries MAX_TRY times
 * before quitting.
 */

int lock(FILE *fp)
{
    int try, fd;
    try = 0;
    lseek(fp, 0, 0);

    /* Get the fileno of the file */

    fd = fileno(fp);
    while (lockf(fd, F_TLOCK, 0) < 0) {
        if (errno == EAGAIN || errno == EACCES) {

            /* Unsuccessful in locking the file try again */

            if (try++ < MAX_TRY) {
                sleep(2);
            } else
                return(0);
        }
        errordialog("FILE IS BUSY TRYING AGAIN .....\\n");
    }
    return(1);
}

/* This function is used to unlock a file */

int unlock(FILE *fp)
{
    int fd;
    fd = fileno(fp);
    lockf(fd, F_ULOCK, 0);
}

/*
* * * * *
* Filename : Load.c *
* * * * *
* Programmed by : N. Sunil Chakravarthy *
* * * * *
* Last updated on : 08.17.94 *
* * * * *
*/

This file contains all the functions for interface with RCS. These functions are called
from the
various callback functions of the graphical user interface.
*/

# include <stdio.h>
# include <ctype.h>
# include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>

```

```

#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <Xm/Scale.h>

/* This structure is designed to hold all the necessary information of the files at each
node.

    name      :      Name of the file.
    rcsfile   :      Name of the rcs file in which the file is stored.
    rcsno     :      This is the logical name which is associated with each file in RCS.
                   This number is used to refer to the files in the rcsfiles. This is
                   a feature of RCS.
    author    :      Author of the file as supplied by the user.
    email     :      E-mail address of the contributor as supplied by the user.
    function  :      Function performed by the program(file) as supplied by the user.
    method    :      Method of performing the function as supplied by the user.
    implementation: Implementation details as provided by the user.
    saved     :      The space saved in storing this file in the library.
    outno     :      The percentage savings in space.
    status    :      Status of the file.
    filesize  :      The original size of the file.

*/

struct file_node {
    char    name[20];
    char    rcsfile[80];
    char    rcsno[2000];
    char    author[20];
    char    email[20];
    char    function[100];
    char    method[100];
    char    implementation[1000];
    float   saved;
    int     outno;
    int     status;
    int     filesize;
    struct file_node *left;
    struct file_node *right;
    struct file_node *parent;
};

/* This structure holds the information of the link names. This is explained in link.c */

extern struct list {
    char    name[20];
    char    filename[20];
    int     type;
    int     distance;
    int     filesize;
    struct list *left;
    struct list *right;
};

extern struct list *listhead;

extern void writelist();
extern void load_list();
extern void printlist();
extern void free_list();
extern void errorDialog();
extern int  file_time();
extern int  lock();
extern void unlock();

extern int  num, tcount;
extern int  scale, toplevel;
extern char *passwd;
struct file_node *head, *point;

char  lib_d[100];
char  src_d[100];

#include "displaygrph.h"

FILE *fp;

```

```

int    add();
void file_write();
int    add_node();
int    delete_node();
void traverse();
int    checkout();
void move_node();
int    check_outall();
int    is_file();
int    is_direct();
int    file_present();
void recheck_in();
void destroy_rcs();
void cin();
int    cout();
void print_tree();
void system_select();
void write_file();
void write_exp();
int    add_exp();
void load_file();
int    space_file();
float space_saved();
struct list *add_list();

char filename[80];
int    gl_test, rcspace;
int    load_time;
float total;
int    size;

/* This function is called at the beginning of the application to load the files of the
library. This function creates a dummy node called the head node (or the root node) and
calls the function add_nodes to load the nodes from the file main.dat. If main.dat does
not exist, this function gives an appropriate error message and terminates the
application. */

load()
{
    struct file_node *cur, *new;
    struct list *listcur;
    char s[100], s1[80], s2[80];
    int done, choice, i;
    strcpy(src_d, "./");

    /* Open Main.dat from the library directory. */

    sprintf(s, "%smain.dat", lib_d);
    fp = fopen(s, "r");
    if (!fp) {
        printf("FILE main.dat does not exist or error opening file ..fatal error
            terminating the program ");
        exit(0);
    }

    /* Creating the dummy initial node. */

    head = (struct file_node *)malloc(sizeof(struct file_node ));
    if (!head) {
        error_dialog("memory allocation error in load ...");
        exit(0);
    }
    strcpy(head->name, "head");
    strcpy(head->rcsno, "HEAD");
    strcpy(head->rcsfile, "NONE");
    strcpy(head->function, "This is the initial dummy node. Attaching a file to this
node
    creates a new branch");
    head->left = NULL;
    head->right = NULL;
    head->parent = NULL;
    head->status = 1;

    add(head, 0);

```



```

        fclose(fp);
    }

    /* This function is called if the system administrator quits the application. Since only
    the system administrator can make changes to the system, the main.dat file is updated
    only for the system administrator. This function opens the file main.dat in the write
    mode and then calls the file_write function. If there is an error in opening main.dat
    file, the application is closed.
    */

void write_file()
{
    char    s[100];

    /* Open main.dat file from the library directory. */

    sprintf(s, "%smain.dat", lib_d);
    fp = fopen(s, "w");
    if (!fp) {
        errordialog("UNABLE TO OPEN BABAI ....");
        exit(0);
    }
    file_write(head);
    fclose(fp);
}

/* This function is called by the load function. This function is used to load all the
files from the main.dat file into the tree structure. This structure is explained
before file write function. The files are loaded in an inorder traversal method. This
is a recursive function.
*/

int    add(struct file_node *child, int dir)
{
    struct file_node *cur;
    char    str[85], str1[80], str2[80], string[2000];
    char    *c, *d, *e;

    if (child) {
        fgets(str, 80, fp);
        c = strtok(str, "\n");
        strcpy(str, c);

        /* Create a node. */

        if (!feof(fp)) {
            cur = (struct file_node *)malloc(sizeof(struct file_node));
            if (strcmp(str, "NULL") == 0) {
                child->left = NULL;
            } else
            {
                /* Load all the information. */

                fgets(str1, 80, fp);
                d = strtok(str1, "\n");
                strcpy(str1, d);
                fgets(str2, 80, fp);
                e = strtok(str2, "\n");
                strcpy(str2, e);
                strcpy(cur->name, str);
                strcpy(cur->rcsfile, str1);
                strcpy(cur->rcsno, str2);
                fgets(str, 80, fp);
                d = strtok(str, "\n");
                strcpy(str, d);
                memset(string, '\0', 2000);
                while (strcmp(str, ".") != 0 && !feof(fp)) {
                    strcat(string, str);
                    fgets(str, 80, fp);
                    d = strtok(str, "\n");
                    strcpy(str, d);
                }
                strcpy(cur->author, string);
                fgets(str, 80, fp);
            }
        }
    }
}

```

```

d = strtok(str, "\n");
strcpy(str, d);
memset(string, '\0', 2000);
while (strcmp(str, ".") != 0 && !feof(fp)) {
    strcat(string, str);
    fgets(str, 80, fp);
    d = strtok(str, "\n");
    strcpy(str, d);
}
strcpy(cur->email, string);
fgets(str, 80, fp);
d = strtok(str, "\n");
strcpy(str, d);
memset(string, '\0', 2000);
while (strcmp(str, ".") != 0 && !feof(fp)) {
    strcat(string, str);
    fgets(str, 80, fp);
    d = strtok(str, "\n");
    strcpy(str, d);
}
strcpy(cur->function, string);
fgets(str, 80, fp);
d = strtok(str, "\n");
strcpy(str, d);
memset(string, '\0', 2000);
while (strcmp(str, ".") != 0 && !feof(fp)) {
    strcat(string, str);
    fgets(str, 80, fp);
    d = strtok(str, "\n");
    strcpy(str, d);
}
strcpy(cur->method, string);
fgets(str, 80, fp);
d = strtok(str, "\n");
strcpy(str, d);
memset(string, '\0', 2000);
while (strcmp(str, ".") != 0 && !feof(fp)) {
    strcat(string, str);
    fgets(str, 80, fp);
    d = strtok(str, "\n");
    strcpy(str, d);
}
strcpy(cur->implementation, string);
fgets(str, 80, fp);
d = strtok(str, "\n");
strcpy(str, d);
memset(string, '\0', 2000);
while (strcmp(str, ".") != 0 && !feof(fp)) {
    strcat(string, str);
    fgets(str, 80, fp);
    d = strtok(str, "\n");
    strcpy(str, d);
}
cur->saved = atof(string);
fgets(str, 80, fp);
d = strtok(str, "\n");
strcpy(str, d);
memset(string, '\0', 2000);
while (strcmp(str, ".") != 0 && !feof(fp)) {
    strcat(string, str);
    fgets(str, 80, fp);
    d = strtok(str, "\n");
    strcpy(str, d);
}
cur->outno = atoi(string);
memset(string, '\0', 2000);
fgets(str, 80, fp);
d = strtok(str, "\n");
strcpy(str, d);
while (strcmp(str, ".") != 0 && !feof(fp)) {
    strcat(string, str);
    fgets(str, 80, fp);
    d = strtok(str, "\n");
    strcpy(str, d);
}
cur->status = atoi(string);
memset(string, '\0', 2000);
fgets(str, 80, fp);

```

```

    d = strtok(str, "\n");
    strcpy(str, d);
    while (strcmp(str, ".") != 0 && !feof(fp)) {
        strcat(string, str);
        fgets(str, 80, fp);
        d = strtok(str, "\n");
        strcpy(str, d);
    }
    cur->filesize = atoi(string);
    if (!feof(fp)) {
        child->left = cur;
        cur->parent = child;
    }
}

fgets(str, 80, fp);
c = strtok(str, "\n");
strcpy(str, c);

/* Load the right node, if present. */
if (!feof(fp)) {
    cur = (struct file_node *)malloc(sizeof(struct file_node));
    if (strcmp(str, "NULL") == 0) {
        child->right = NULL;
    } else
    {
        fgets(str1, 80, fp);
        d = strtok(str1, "\n");
        strcpy(str1, d);
        fgets(str2, 80, fp);
        e = strtok(str2, "\n");
        strcpy(str2, e);
        strcpy(cur->name, str);
        strcpy(cur->rcsfile, str1);
        strcpy(cur->rcsno, str2);
        fgets(str, 80, fp);
        d = strtok(str, "\n");
        strcpy(str, d);
        memset(string, '\0', 2000);
        while (strcmp(str, ".") != 0 &&
            !feof(fp)) {
            strcat(string, str);
            fgets(str, 80, fp);
            d = strtok(str, "\n");
            strcpy(str, d);
        }
        strcpy(cur->author, string);
        fgets(str, 80, fp);
        d = strtok(str, "\n");
        strcpy(str, d);
        memset(string, '\0', 2000);
        while (strcmp(str, ".") != 0 &&
            !feof(fp)) {
            strcat(string, str);
            fgets(str, 80, fp);
            d = strtok(str, "\n");
            strcpy(str, d);
        }
        strcpy(cur->email, string);
        fgets(str, 80, fp);
        d = strtok(str, "\n");
        strcpy(str, d);
        memset(string, '\0', 2000);
        while (strcmp(str, ".") != 0 &&
            !feof(fp)) {
            strcat(string, str);
            fgets(str, 80, fp);
            d = strtok(str, "\n");
            strcpy(str, d);
        }
        strcpy(cur->function, string);
        fgets(str, 80, fp);
        d = strtok(str, "\n");
        strcpy(str, d);
        memset(string, '\0', 2000);
        while (strcmp(str, ".") != 0 &&
            !feof(fp)) {

```

```

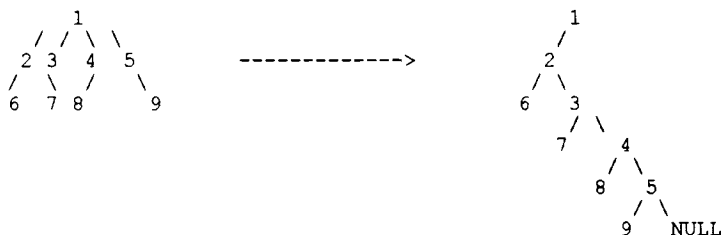
        strcat(string, str);
        fgets(str, 80, fp);
        d = strtok(str, "\n");
        strcpy(str, d);
    }
    strcpy(cur->method, string);
    fgets(str, 80, fp);
    d = strtok(str, "\n");
    strcpy(str, d);
    memset(string, '\0', 2000);
    while (strcmp(str, ".") != 0 &&
        !feof(fp)) {
        strcat(string, str);
        fgets(str, 80, fp);
        d = strtok(str, "\n");
        strcpy(str, d);
    }
    strcpy(cur->implementation, string);
    fgets(str, 80, fp);
    d = strtok(str, "\n");
    strcpy(str, d);
    memset(string, '\0', 2000);
    while (strcmp(str, ".") != 0 &&
        !feof(fp)) {
        strcat(string, str);
        fgets(str, 80, fp);
        d = strtok(str, "\n");
        strcpy(str, d);
    }
    cur->saved = atof(string);
    fgets(str, 80, fp);
    d = strtok(str, "\n");
    strcpy(str, d);
    memset(string, '\0', 2000);
    while (strcmp(str, ".") != 0 &&
        !feof(fp)) {
        strcat(string, str);
        fgets(str, 80, fp);
        d = strtok(str, "\n");
        strcpy(str, d);
    }
    cur->outno = atoi(string);
    memset(string, '\0', 2000);
    fgets(str, 80, fp);
    d = strtok(str, "\n");
    strcpy(str, d);
    while (strcmp(str, ".") != 0 &&
        !feof(fp)) {
        strcat(string, str);
        fgets(str, 80, fp);
        d = strtok(str, "\n");
        strcpy(str, d);
    }
    cur->status = atoi(string);
    memset(string, '\0', 2000);
    fgets(str, 80, fp);
    d = strtok(str, "\n");
    strcpy(str, d);
    while (strcmp(str, ".") != 0 &&
        !feof(fp)) {
        strcat(string, str);
        fgets(str, 80, fp);
        d = strtok(str, "\n");
        strcpy(str, d);
    }
    cur->filesize = atoi(string);
    if (!feof(fp)) {
        child->right = cur;
        cur->parent = child;
    }
}
    }
    add(child->left, 0);
    add(child->right, 1);
} else
    return(1);
}

```

- ```

/*
* This function is called from the write_file on trying to quit the application. This
function is called only if the user password is that of the library administrator as
only the library administrator is permitted to make changes to the system. The
main.dat file is updated.
* The structure of storing the tree structure of the library files is as follows.
* First the tree with an unlimited number of children is converted into a binary tree as
follows

```



This tree is then stored in an inorder traversal order.

```

1
2
NULL
6
3
NULL
NULL
7
4
NULL
NULL
8
5
NULL
NULL
9
NULL

```

```

*/

```

```

void file_write(struct file_node *cur)
{
 char str[80];
 if (cur) {

 /* Write the left node and its information, if present. */

 if (cur->left && cur->left->status) {
 fputs(cur->left->name, fp);
 fputs("\n", fp);
 fputs(cur->left->rcsfile, fp);
 fputs("\n", fp);
 fputs(cur->left->rcsno, fp);
 fputs("\n", fp);
 fputs(cur->left->author, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 fputs(cur->left->email, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 fputs(cur->left->function, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 fputs(cur->left->method, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 fputs(cur->left->implementation, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 sprintf(str, "%f", cur->left->saved);
 fputs(str, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 sprintf(str, "%d", cur->left->outno);
 fputs(str, fp);

```

```

 fputs("\n", fp);
 fputs(".\n", fp);
 sprintf(str, "%d", cur->left->status);
 fputs(str, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 sprintf(str, "%d", cur->left->filesize);
 fputs(str, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 } else {
 if (!cur->left) {
 fputs("NULL", fp);
 fputs("\n", fp);
 }
 }

 /* Write the right node and its information, if present. */
 if (cur->right && cur->right->status) {
 fputs(cur->right->name, fp);
 fputs("\n", fp);
 fputs(cur->right->rcsfile, fp);
 fputs("\n", fp);
 fputs(cur->right->rcsno, fp);
 fputs("\n", fp);
 fputs(cur->right->author, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 fputs(cur->right->email, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 fputs(cur->right->function, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 fputs(cur->right->method, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 fputs(cur->right->implementation, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 sprintf(str, "%f", cur->right->saved);
 fputs(str, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 sprintf(str, "%d", cur->right->outno);
 fputs(str, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 sprintf(str, "%d", cur->right->status);
 fputs(str, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 sprintf(str, "%d", cur->right->filesize);
 fputs(str, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 } else {
 if (!cur->right) {
 fputs("NULL", fp);
 fputs("\n", fp);
 }
 }
 file_write(cur->left);
 file_write(cur->right);
}

/*
 * This function can be used to print the files in the tree in an inorder traversal. This
 * function is not used in the interface. This can be used during debugging after
 * adapting the printf in the function to print the necessary information. This is a
 * recursive function.
 */

```

```
void print_tree(struct file_node *cur)
```

```

{
 if (cur) {
 if (cur->left) {
 printf("file name : %s logical name: %s rcsfile: %s \n",
 cur->left->name, cur->left->rcsno, cur->left->rcsfile);
 } else {
 fputs("NULL", fp);
 fputs("\n", fp);
 }
 if (cur->right) {
 printf("file name : %s logical name: %s rcsfile: %s \n",
 cur->right->name, cur->right->rcsno, cur->right->rcsfile);
 } else {
 fputs("NULL", fp);
 fputs("\n", fp);
 }
 print_tree(cur->left);
 print_tree(cur->right);
 }
}

/*
 * This function is used to delete a node from the library. The parameter passed is the
 * name of the file to be deleted. The steps involved in doing so are the following:
 * a. Locate the file in the tree and return a pointer to the node. If the node is
 * not located, give a warning and return.
 * b. Update the tree after deleting the node.
 * c. Update the RCS file by deleting the file from it. This involves checking out
 * all the files and checking them back in in the desired order.
 */

int delete_node(char s[80])
{
 struct file_node *cur, *temp;

 /* Locate the node. */

 traverse(head, s);
 cur = point;
 if (!cur) {

 /* Node is not found, give an error message. */

 errordialog("node not found \n");
 return(1);
 }

 /* If the file is not an experimental file, check out all the files in the
 corresponding RCS file.
 */

 if (cur->status) {
 check_outall(head->left);
 destroy_rcs(head->left);
 }

 /* Remove the node from the tree. */

 /* If the node has any children, link all of them to the parent of this node.*/

 if (cur->left) {
 temp = cur->left;

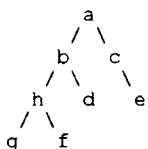
 /* Connect the child of the node to its parent in the correct direction
 (left or right) */

 if (strcmp(cur->parent->right->name, cur->name) == 0)
 cur->parent->right = temp;
 else
 cur->parent->left = temp;
 temp->parent = cur->parent;

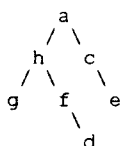
 /*
 Connect all children of this node to its parent before its previous
 children.
 */
 }
}

```

For example,



If b has to be deleted, the resulting tree should be



\*/

```

while (temp->right != NULL)
 temp = temp->right;
temp->right = cur->right;
if (cur->right) {
 cur->right->parent = temp;
}
} else {
 if (strcmp(cur->parent->right->name, cur->name) == 0)
 cur->parent->right = cur->right;
 else
 cur->parent->left = cur->right;
 if (cur->right)
 cur->right->parent = cur->parent;
}

/* If the node is not an experimental node, recheck-in the RCS file in the right
order. */

if (cur->status)
 recheck_in(head->left);
}

/* This function is used to check-in a file into the library. The various steps involved
* are as follows:
* a. Locate the parent node from its name passed as the second argument.
* b. Link the node (argument 1) as a child of the parent in the DAG.
* c. Call the cin function to actually check-in the file as an RCS file.
*/

int
add_node(struct file_node *newnode, char s[80])
{
 struct file_node *cur, *temp, *templ;
 int count, fcount = 0;
 point = NULL;

 /* Locate the parent node. */

 traverse(head, s);
 cur = point;
 if (!cur) {
 /* Parent not found, give error message and return. */
 error_dialog("node not found \n");
 return(1);
 }
 templ = cur;

 /* Check to see if the depth is more than 8 levels. If so, give an error message
and return.
RCS gives a core dump if the string length of the version number exceeds a

```



```

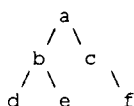
certain length. This is exceeded in 9 levels. */
while (templ != head) {
 while (strcmp(templ->parent->left->name, templ->name) !=
 0)
 templ = templ->parent;
 templ = templ->parent;
 fcount++;
}
point = NULL;

/* Check to see if another file with the same name exists in the library. If so,
return after giving an error message.
*/

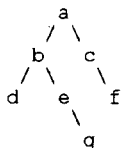
traverse(head, newnode->name);
if (point) {
 errorialog("file with same name present, checkin after changing the name
 ..");
 return(1);
}
if (fcount > 7) {
 errorialog("The number of levels exceeds 7, start a new branch from
 head");
 return(1);
}
count = 2;

/* Connect the node as the rightmost child of its parent.
For example,

```



If node g is to be connected to node a, the resulting tree is as follows.



```

*/
if (cur->left) {
 temp = cur->left;
 while (temp->right != NULL) {
 count++;
 temp = temp->right;
 }
 temp->right = newnode;
 newnode->parent = temp;
 strcpy(newnode->rscfile, cur->rscfile);
} else {
 cur->left = newnode;
 newnode->parent = cur;
 strcpy(newnode->rscfile, cur->rscfile);
}

/* Gl_test is the count of the number of children above the connected node +2. In
the above example, it is 4(b,e). This is used in the cin function. */

gl_test = count;

/* Call the function to check in the file into the corresponding RCS file. */
cin(newnode);
}

```

```

/*
* This function is used to locate a node in the tree. The global pointer "Point" is set
to point to the node if found. This function is used in the other interface functions
in this file.

```

```

* This is a recursive function and employs a preorder traversal technique.
*/

void traverse(struct file_node *cur, char s[80])
{
 if (cur) {
 if (strcmp(cur->name, s) == 0)
 point = cur;
 traverse(cur->left, s);
 traverse(cur->right, s);
 }
}

/*
* This function is used to move a node. The procedure is as follows:
* a. Locate the node to be moved.
* b. Create a new node and copy the information from the above node.
* c. Delete the located node.
* d. Check-in the created node as a child of the desired parent.
* This function is called from the move option callback in the user interface.
*/

void move_node(char s[80], char s1[80])
{
 struct file_node *new, *cur;
 traverse(head->left, s);
 if (point) {

 /* Located the node, now create the new node */

 new = (struct file_node *)malloc(sizeof(struct file_node));
 if (!new) {
 error_dialog("allocation error");
 exit(0);
 }
 new->left = NULL;
 new->right = NULL;
 new->parent = NULL;

 /* Copy the information. */

 strcpy(new->name, point->name);
 strcpy(new->racsno, point->racsno);
 strcpy(new->author, point->author);
 strcpy(new->email, point->email);
 strcpy(new->function, point->function);
 strcpy(new->method, point->method);
 strcpy(new->implementation, point->implementation);
 memset(new->racsno, '\0', 2000);
 new->saved = 0;
 new->outno = 0;
 new->status = point->status;

 /* Delete the original node. */

 delete_node(s);
 strcpy(src_d, lib_d);

 /* Connect the copy to the new parent. */

 add_node(new, s1);
 }
}

/*
* This function is used to check-out a file from the library into the current directory
* of the user. The various steps involved are as follows.
* a. Locate the desired node from its name(argument1).
* b. If the status of the node is experimental, the file exists in its original
* format in the library directory. All that needs to be done is to copy the file
* into the current directory of the user.
* c. If the status is not experimental, call function cout to checkout the file
* from the RCS file.
* d. Record the check-out time in the cout.dat file. This is used in plotting the
* frequency of the check-out graph.
*/

```

```

int checkout(char s[80])
{
 char str[100];
 time_t * tloc, t;
 tloc = (time_t *)malloc(sizeof(time_t));
 point = NULL;
 traverse(head->left, s);
 if (point) {
 if (!point->status) {
 sprintf(str, "cp %s%s %s%s 2>%serrlog", lib_d, point->name,
 src_d, point->name, lib_d);
 system(str);
 return(0);
 }

 /* First lock the cout.dat file to prevent possible corruption due to
 simultaneous updates. */

 sprintf(str, "%scout.dat", lib_d);
 fp = fopen(str, "r+");
 if (!fp) {
 errordialog("UNABLE TO OPEN cout.dat file");
 } else {
 if (lock(fp)) {
 fseek(fp, 0, 2);
 if (!tloc)
 errordialog("memory allocation problem...");
 else
 t = time(tloc);
 sprintf(str, "%s %d\n", point->name, t);

 /* Store the time of checkout. */

 fputs(str, fp);
 unlock(fp);
 fclose(fp);
 } else
 errordialog("UNABLE TO LOCK checkout file .. checkout not
 recorded");
 }

 cout(point);
 } else
 errordialog("NODE NOT FOUND \n");
}

/*
 * This function is used to check out the file from its RCS file.
 * The various steps involved are:
 * a. Retrieve the name of the RCS file from the information in the node.
 * b. Retrieve the RCS version number from the information in the node.
 * c. Check-out the file using the above information.
 * d. Copy the file into the user directory after renaming (if necessary) to its
 * name in the information. Renaming is necessary because if two files (say a.c
 * and b.c) are checked into a RCS file called l.c,v for instance, then when a
 * file (a.c) is checked out from l.c,v it's contents are placed in l.c. These
 * contents must be then copied
 * into the file a.c.
 */
int cout(struct file_node *cur)
{
 char str[200], str1[200];
 char *c;
 if (cur) {
 if (cur->status) {
 strcpy(str, cur->rcsfile);
 c = strtok(str, ",");
 strcpy(str1, c);
 sprintf(str, "co -u -f -r%s %s%s %s%s 2>%serrlog",
 cur->rcsno, lib_d, cur->rcsfile, lib_d, str1, lib_d);
 system(str);
 strcpy(str, cur->rcsfile);
 c = strtok(str, ",");
 strcpy(str, c);
 sprintf(str1, "%s%s", src_d, cur->name);
 if (is_file(str1)) {
 sprintf(str1, "chmod 777 %s%s 2>%serrlog",

```

```

 src_d, cur->name, lib_d);
 system(str1);
 }
 sprintf(str1, "cp %s%s %s%s 2>%serrlog", lib_d, str,
 src_d, cur->name, lib_d);
 system(str1);
} else {
 sprintf(str, "cp %s%s %s%s 2>%serrlog", lib_d, cur->name,
 src_d, cur->name, lib_d);
 system(str);
}
}
}

/*
 * This function is used to recursively check-out all the files in the library. This
 * function is called if there is a change in the library structure. A change in the tree
 * is reflected in the RCS file only if all files are checked-out and checked-in again in
 * the right order.
 */

int check_outall(struct file_node *cur)
{
 char str[200], str1[200];
 char *c;
 strcpy(str, cur->rscfile);
 c = strtok(str, ",");
 strcpy(str1, c);

 if (cur) {
 if (cur->status) {
 sprintf(str, "%s%s", lib_d, cur->rscfile);

 /* Check if the RCS file is present. */

 if (is_file(str)) {
 sprintf(str, "co -u -f -r%s %s%s %s%s 2>%serrlog",
 cur->rscno, lib_d, cur->rscfile, lib_d,
 str1, lib_d);

 /* Check out from the RCS File. */

 system(str);
 strcpy(str, cur->rscfile);
 c = strtok(str, ",");
 strcpy(str, c);
 sprintf(str1, "%s%s", lib_d, cur->name);

 /* Rename if necessary. */

 if (is_file(str1)) {
 sprintf(str1, "chmod 777 %s%s 2>%serrlog",
 lib_d, cur->name, lib_d);
 system(str1);
 }
 sprintf(str1, "cp %s%s %s%s 2>%serrlog",
 lib_d, str, lib_d, cur->name, lib_d);
 system(str1);
 }
 check_outall(cur->left);
 check_outall(cur->right);
 }
 }
}

/* This function is used to check if a file is present in the library directory. */

int file_present(char file[])
{
 char str[200];
 sprintf(str, "test -r %s 2>%serrlog", file, lib_d);
 if (system(str)) {
 errordialog("FILE NOT PRESENT OR READABLE.....\n");
 return 0;
 } else
 return 1;
}

```

```

}
/* This function is used to check if a directory is valid or accessible. */
int is_direct(char direct[])
{
 char str[200];
 sprintf(str, "test -d %s2>%serrlog", direct, lib_d);
 if (system(str)) {
 errorialog("DIRECTORY NOT ACCESSIBLE OR NOT PRESENT\\n");
 return 0;
 } else
 return 1;
}

/* This function is used to check if a file is present or not. The path of the file is
part of the argument.
*/

int is_file(char file[])
{
 struct stat s_buf;
 if (stat(file, &s_buf) == -1)
 return 0;
 else
 return 1;
}

/* This function is used to destroy the given RCS file. This function is called after
checking-out all the files from the RCS file in order to check-in again.
*/

void destroy_rcs(struct file_node *cur)
{
 char str[200];
 while (cur != NULL) {
 sprintf(str, "chmod 777 %s%s 2>%serrlog ", lib_d, cur->rscfile,
 lib_d);
 system(str);
 sprintf(str, "rm %s%s 2>%serrlog", lib_d, cur->rscfile, lib_d);
 system(str);
 cur = cur->right;
 }
}

/* This function is used to check-in a file as a RCS file. The various steps involved are:
* a. Get the RCS file name of the parent. If the parent is the head, create a new
* RCS file.
* b. Get the RCS number of the new file by calculating as follows:
*
* a(1.1)
* / \
* b c
* / \
* d e
* \
* f
*
* the RCS number of f will be 1.1.g1_test. As g1_test=4 (see add_node for
* g1_test)it will be 1.1.4.
* c. Check in the new file with the logical name and revision number.
*/

void cin(struct file_node *cur)
{
 struct file_node *temp;
 char str[800], str1[800], name[100];
 char *c;
 if (cur) {
 if (cur->status) {
 temp = cur;

 /* Get the parent of the node. */

 while (strcmp(cur->parent->left->name, cur->name)

```

```

!= 0)
 cur = cur->parent;
if (cur->parent != head) {
 /* Get the RCS filename as the parent is not the head */
 strcpy(str, cur->parent->rscfile);
 c = strtok(str, ",");
 strcpy(str, c);
 strcpy(name, str);
 sprintf(str1, "%s%s", lib_d, str);

 /* If there is a file in the library directory with the same
 name make sure that it can be overwritten. */
 if (is_file(str1)) {
 sprintf(str1, "chmod 777 %s%s 2>%serrlog",
 lib_d, str, lib_d);
 system(str1);
 }

 /* Get the file to be checked-in into the library directory,
 from the user directory. */
 sprintf(str1, "cp %s%s %s%s 2>%serrlog",
 src_d, temp->name, lib_d, str, lib_d);
 system(str1);

 /* Record the space occupied by the RCS file before
 check-in. */
 rcspace = space_file(cur->rscfile, lib_d);

 /* Check-in with the right version number.
 Case 1 is when there are no children to its parent.
 Case 2 is explained in the header to this function. */
 if (strcmp(temp->parent->left->name, temp->name)
 == 0) {
 sprintf(str, "ci -f -t%smsg -m%smsg -n%s -r%s.l
 %s%s %s%s2>%serrlog", lib_d, lib_d,
 temp->rscno, temp->parent->rscno, lib_d,
 cur->parent->rscfile, lib_d, name, lib_d);
 system(str);
 strcpy(temp->rscfile, cur->parent->rscfile);
 } else {
 sprintf(str, "ci -f -t%smsg -m%smsg -n%s -r%s.%d
 %s%s%s 2>%serrlog", lib_d, lib_d,
 temp->rscno, cur->parent->rscno, gl_test,
 lib_d, cur->parent->rscfile, lib_d, name,
 lib_d);
 system(str);
 strcpy(temp->rscfile, cur->parent->rscfile);
 }
} else {
 /* Create a new RCS file. */
 rcspace = 0;
 cur = temp;
 sprintf(str1, "%s%s", lib_d, cur->name);
 if (is_file(str1)) {
 sprintf(str1, "chmod 777 %s%s 2>%serrlog",
 lib_d, cur->name, lib_d);
 system(str1);
 }
 sprintf(str1, "cp %s%s %s%s 2>%serrlog",
 src_d, cur->name, lib_d, cur->name, lib_d);
 system(str1);
 sprintf(str, "ci -f -u -t%smsg -m%smsg -n%s %s%s
 2>%serrlog", lib_d, lib_d, cur->rscno, lib_d,
 cur->name, lib_d);
 system(str);
 sprintf(cur->rscfile, "%s,v", cur->name);
}
total = 0.0;
size = 0;

```

```

 /* Record the space of the RCS file after the check-in process.
 Calculate the space saved. */
 temp->saved = space_saved(temp->name, temp->rscfile);
 temp->outno = total;
 temp->filesize = size;
 printf("SPACE SAVED IN CHECKING IN THIS FILE IS ...%f , total =
 %f\n",temp->saved, total);
 }
}

/* This function is used to recheck-in the files once they are all checked-out from the
RCS file. This function is called after the function check_outall is called and some
changes are made to the library structure. It is a recursive function and all files are
checked-in using the information in the nodes. Same techniques as in cin function are
employed here.
*/

void recheck_in(struct file_node *cur)
{
 struct file_node *temp, *point;
 char str[800], str1[800], name[100];
 int count;
 char *c;
 if (cur) {
 temp = cur;
 if (cur->status) {
 while (strcmp(cur->parent->left->name, cur->name)
 != 0)
 cur = cur->parent;
 point = cur;
 if (cur->parent != head) {
 strcpy(str, cur->parent->rscfile);
 c = strtok(str, ",");
 strcpy(str, c);
 strcpy(name, c);
 sprintf(str1, "%s%s", lib_d, str);
 if (is_file(str1)) {
 sprintf(str1, "chmod 777 %s%s 2>%serrlog",
 lib_d, str, lib_d);
 system(str1);
 }
 sprintf(str1, "cp %s%s %s%s 2>%serrlog",
 lib_d, temp->name, lib_d, str, lib_d);
 system(str1);
 sleep(1);
 if (strcmp(temp->parent->left->name, temp->name)
 == 0) {
 sprintf(str, "ci -f -t%smsg -m%smsg -n%s -r%s.1
 %s%s %s%s 2>%serrlog",lib_d, lib_d, temp->rscno,
 temp->parent->rscno,lib_d,
 cur->parent->rscfile,
 lib_d, name, lib_d);
 system(str);
 } else {
 count = 2;
 while (strcmp(point->name, temp->name)
 != 0) {
 count++;
 point = point->right;
 }
 sprintf(str, "ci -f -t%smsg -m%smsg -n%s -r%s.%d
 %s%s%s 2>%serrlog",lib_d, lib_d,
 temp->rscno, cur->parent->rscno, count,
 lib_d,
 cur->parent->rscfile, lib_d, name, lib_d);
 system(str);
 }
 strcpy(temp->rscfile, cur->parent->rscfile);
 } else {
 cur = temp;
 sprintf(str, "ci -f -t%smsg -m%smsg -n%s %s%s 2>%serrlog",
 lib_d, lib_d, cur->rscno, lib_d, cur->name,
 lib_d);
 system(str);
 sprintf(cur->rscfile, "%s,v", cur->name);
 }
 }
 }
}

```

```

 }
 recheck_in(temp->left);
 recheck_in(temp->right);
}

/*
 * This function is used to select a likely candidate for the parent of any given node.
 * This function is called from the callback of "position selected by the system" choice
 * in the user interface. The various steps involved are:
 * a. Retrieve the RCS file at each node in the tree.
 * b. Compare the given file with the RCS file using RCSDIFF.
 * c. Set the filename variable to the name of the file giving the least
 * difference.
 */
void system_select(struct file_node *cur, char s[80])
{
 struct file_node *templ;
 char str[200], str1[200];
 char *c;
 int i, fcount = 0;
 if (cur) {
 /* Update the scale in the working dialog. */
 num++;
 if ((num % 5) == 0) {
 if (num * 100 / tcount > 1)
 XmScaleSetValue(scale, (int)(num * 100 /
 tcount));
 }

 /* Synchronize the display so that the changes to scale is updated on
 the screen. */
 XSync(XtDisplay(toplevel), False);
 sprintf(str, "%s%s", lib_d, cur->rscfile);

 /* Check to see if the RCS file is in the library directory and that the
 current node is not an experimental node. */
 if (is_file(str) && cur->status) {
 strcpy(str, cur->rscfile);
 c = strtok(str, "");
 strcpy(str, c);
 sprintf(str1, "%s%s", lib_d, str);
 if (is_file(str1)) {
 sprintf(str1, "chmod 777 %s%s 2>%serrlog",
 lib_d, str, lib_d);
 system(str1);
 }

 /* Copy the given file into the library directory. */
 sprintf(str1, "cp %s%s %s%s 2>%serrlog", src_d,
 s, lib_d, str, lib_d);
 system(str1);

 /* Compare the given file with the current node (file) in the RCS
 file. The number of differing lines is counted by "wc" and the
 output is stored in file s of the library directory.
 */
 sprintf(str1, "rcsdiff -r%s %s%s 2>%serrlog|wc -l >%ss",
 cur->rscno, lib_d, str, lib_d, lib_d, passwd);
 system(str1);

 /* Open the file s and retrieve the above count and delete the
 file. */
 sprintf(str, "%s%s", lib_d, passwd);
 fp = fopen(str, "r");
 fgets(str1, 80, fp);
 fclose(fp);
 sprintf(str, "rm %s%s 2>%serrlog", lib_d, passwd,
 lib_d);
 system(str);
 }
 }
}

```



```

 /* Compare to see if it is the least count so far encountered.
 If so, check to see if 7 levels are not exceeded and note the
 current node as the selected parent. */
 i = atoi(str1);
 if (i < gl_test) {
 templ = cur;
 while (templ != head) {
 while (strcmp(templ->parent->left->name,
 templ->name) != 0)
 templ = templ->parent;
 templ = templ->parent;
 fcount++;
 }
 if (fcount < 7) {
 gl_test = i;
 strcpy(filename, cur->name);
 }
 }
 system_select(cur->left, s);
 system_select(cur->right, s);
 }
}

/* This function is used to get the disk space of the given file in the given
 directory. The method used is as follows. Do a ls -l of the file in the
 given directory and redirect the output to a file s in the library directory.
 Retrieve the space from the corresponding field in the file s.
*/
int space_file(char s[80], char direct[80])
{
 char str[200], str1[200];
 int i, j, k;
 sprintf(str, "ls -l %s%s >%s", direct, s, lib_d);
 system(str);
 sprintf(str, "%s", lib_d);
 fp = fopen(str, "r");
 fgets(str, 80, fp);
 fclose(fp);
 i = 0;
 j = 0;
 k = 0;
 for (j = 0; j < 5; j++) {
 memset(str1, '\0', 80);
 while (str[i] != ' ') {
 str1[k] = str[i];
 i++;
 k++;
 }
 k = 0;
 while (str[i] == ' ')
 i++;
 }
 printf("string .. %s\n", str1);
 i = atoi(str1);
 return i;
}

/* This function is used to calculate the disk space saved after a file is
 checked-in. The original space of the RCS file before check-in is noted in
 cin function. This is the variable rcsfile. The space of the given file
 and the new rcsfile (with the file checked in) is calculated using
 saved= original space of file - new RCS file + old RCS file space.
*/

float space_saved(char s[80], char rcs[80])
{
 int space, spacel;
 float saved;
 space = space_file(s, src_d);
 spacel = space_file(rcs, lib_d);
 saved = space - spacel + rcspace;
 total = saved;
 size = space;
}

```

```

 if (space != 0)
 saved = saved / space * 100;
 else
 errordialog("not able to calculate original file disk usage ");
 return saved;
}

struct list *add_list(struct list *cur, struct file_node *temp)
{
 struct list *add;
 if (temp) {
 add = (struct list *)malloc(sizeof(struct list));
 if (!add) {
 printf("memory allocation problem ...\n");
 exit(0);
 }
 strcpy(add->name, temp->name);
 add->filename[0] = '\0';
 add->type = 0;
 add->distance = 0;
 cur->right = add;
 add->right = NULL;
 add->left = cur;
 add->filesize = temp->filesize;
 cur = add_list(add, temp->left);
 cur = add_list(cur, temp->right);
 }
 return(cur);
}

/* This function is similar to function "add". This function handles Experimental files.
*/

void load_file()
{
 FILE * fp;
 char str[80];
 struct file_node *cur;
 char str1[80], str2[80], string[2000], connect[100];
 char *c, *d, *e;

 sprintf(str, "%sExperimental.dat", lib_d);
 fp = fopen(str, "r");
 if (!fp) {
 errordialog("FILE OPENING ERROR IN EXPERIMENTAL FILE.");
 return;
 }
 fgets(str, 80, fp);
 c = strtok(str, "\n");
 strcpy(str, c);
 strcpy(connect, str);
 while (!feof(fp)) {
 cur = (struct file_node *)malloc(sizeof(struct file_node));
 fgets(str, 80, fp);
 d = strtok(str, "\n");
 strcpy(str, d);
 fgets(str1, 80, fp);
 d = strtok(str1, "\n");
 strcpy(str1, d);
 fgets(str2, 80, fp);
 e = strtok(str2, "\n");
 strcpy(str2, e);
 strcpy(cur->name, str);
 strcpy(cur->rcsfile, str1);
 strcpy(cur->rcsno, str2);

 fgets(str, 80, fp);
 d = strtok(str, "\n");
 strcpy(str, d);
 memset(string, '\0', 2000);
 while (strcmp(str, ".") != 0 && !feof(fp)) {
 strcat(string, str);
 fgets(str, 80, fp);
 d = strtok(str, "\n");
 strcpy(str, d);
 }
 }
}

```

```

strcpy(cur->author, string);
fgets(str, 80, fp);
d = strtok(str, "\n");
strcpy(str, d);
memset(string, '\0', 2000);
while (strcmp(str, ".") != 0 && !feof(fp)) {
 strcat(string, str);
 fgets(str, 80, fp);
 d = strtok(str, "\n");
 strcpy(str, d);
}
strcpy(cur->email, string);
fgets(str, 80, fp);
d = strtok(str, "\n");
strcpy(str, d);
memset(string, '\0', 2000);
while (strcmp(str, ".") != 0 && !feof(fp)) {
 strcat(string, str);
 fgets(str, 80, fp);
 d = strtok(str, "\n");
 strcpy(str, d);
}
strcpy(cur->function, string);
fgets(str, 80, fp);
d = strtok(str, "\n");
strcpy(str, d);
memset(string, '\0', 2000);
while (strcmp(str, ".") != 0 && !feof(fp)) {
 strcat(string, str);
 fgets(str, 80, fp);
 d = strtok(str, "\n");
 strcpy(str, d);
}
strcpy(cur->method, string);
fgets(str, 80, fp);
d = strtok(str, "\n");
strcpy(str, d);
memset(string, '\0', 2000);
while (strcmp(str, ".") != 0 && !feof(fp)) {
 strcat(string, str);
 fgets(str, 80, fp);
 d = strtok(str, "\n");
 strcpy(str, d);
}
strcpy(cur->implementation, string);
fgets(str, 80, fp);
d = strtok(str, "\n");
strcpy(str, d);
memset(string, '\0', 2000);
while (strcmp(str, ".") != 0 && !feof(fp)) {
 strcat(string, str);
 fgets(str, 80, fp);
 d = strtok(str, "\n");
 strcpy(str, d);
}
cur->saved = atof(string);
fgets(str, 80, fp);
d = strtok(str, "\n");
strcpy(str, d);
memset(string, '\0', 2000);
while (strcmp(str, ".") != 0 && !feof(fp)) {
 strcat(string, str);
 fgets(str, 80, fp);
 d = strtok(str, "\n");
 strcpy(str, d);
}
cur->outno = atoi(string);
memset(string, '\0', 2000);
fgets(str, 80, fp);
d = strtok(str, "\n");
strcpy(str, d);
while (strcmp(str, ".") != 0 && !feof(fp)) {
 strcat(string, str);
 fgets(str, 80, fp);
 d = strtok(str, "\n");
 strcpy(str, d);
}
cur->status = atoi(string);

```

```

 if (!feof(fp))
 add_exp(cur, connect);
 fgets(str, 80, fp);
 c = strtok(str, "\n");
 strcpy(str, c);
 strcpy(connect, str);
 }
 fclose(fp);
}

/* This function is similar to the add_node function. This function handles Experimental
files. Refer to the add_node function for further details. */

int add_exp(struct file_node *newnode, char s[80])
{
 struct file_node *cur, *temp;
 int mcount;
 point = NULL;
 traverse(head, s);
 cur = point;
 if (!cur) {
 return(1);
 }
 count = 2;
 if (cur->left) {
 temp = cur->left;
 while (temp->right != NULL) {
 count++;
 temp = temp->right;
 }
 temp->right = newnode;
 newnode->parent = temp;
 strcpy(newnode->rscsfile, cur->rscsfile);
 } else {
 cur->left = newnode;
 newnode->parent = cur;
 strcpy(newnode->rscsfile, cur->rscsfile);
 }
 gl_test = count;
}

/* This function is similar to the file write function. This function handles Experimental
files. Refer to the file_write function for further details. */

void write_exp(struct file_node *cur, char con[80])
{
 char s[100], str[100];
 sprintf(s, "%sExperimental.dat", lib_d);
 fp = fopen(s, "r+");
 if (!fp) {
 errordialog("UNABLE TO OPEN experimental file");
 } else {
 if (lock(fp)) {
 fseek(fp, 0, 2);
 if (cur) {
 fputs(con, fp);
 fputs("\n", fp);
 fputs(cur->name, fp);
 fputs("\n", fp);
 fputs(cur->rscsfile, fp);
 fputs("\n", fp);
 fputs(cur->rscsno, fp);
 fputs("\n", fp);
 fputs(cur->author, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 fputs(cur->email, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 fputs(cur->function, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 fputs(cur->method, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 fputs(cur->implementation, fp);
 }
 }
 }
}

```

```

 fputs("\n", fp);
 fputs(".\n", fp);
 sprintf(str, "%f", cur->saved);
 fputs(str, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 sprintf(str, "%d", cur->outno);
 fputs(str, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 sprintf(str, "%d", cur->status);
 fputs(str, fp);
 fputs("\n", fp);
 fputs(".\n", fp);
 }
 } else
 errordialog("FILE LOCKING ERROR IN EXPERIMENT FILE ");
 fclose(fp);
}

/*
 * * * * *
 * Filename : List.h
 *
 * Programmed by : N. Sunil Chakravarthy
 *
 * Last updated on : 08.17.94
 * * * * *
*/
#include <stdio.h>
struct list{
 char name[20];
 char filename[20];
 int type;
 int distance;
 int filesize;
 struct list * left;
 struct list * right;
};

struct list *listhead;
void printlist();
void writelist();
void free_list();

/*
 * * * * *
 * Filename : List.c
 *
 * Programmed by : N. Sunil Chakravarthy
 *
 * Last updated on : 08.17.94
 * * * * *
*/

The routines to maintain the list of logical links is contained in this file.
*/

#include <stdio.h>
#include "list.h"

extern char lib_d[100];

/* This function is used to load the list of logical links from
 * the file in the library directory.
 */

void load_list()
{
 FILE * fp;
 struct list *prev, *k;
 char s[1], p[20], q[20], r[20], t[20], str[100];

```

```

int j = 0;
sprintf(str, "%sfff.c", lib_d);
fp = fopen(str, "r+");

/* Create the head node of the linked list */

listhead = (struct list *)malloc(sizeof(struct list));
listhead->left = NULL;
listhead->right = NULL;
prev = listhead;
fgets(s, 2, fp);

/* Read the name of link, the filename to which it is connected, the status of the
file, and the distance.
*/

while (!feof(fp)) {
 j = 0;
 memset(p, '\0', 20);
 while (s[0] != ' ' && !feof(fp)) {
 strcat(p, s);
 fgets(s, 2, fp);
 }

 j = 0;
 memset(q, '\0', 20);
 if (!feof(fp))
 fgets(s, 2, fp);
 while (s[0] != ' ' && !feof(fp)) {
 strcat(q, s);
 fgets(s, 2, fp);
 }
 memset(r, '\0', 20);
 if (!feof(fp))
 fgets(s, 2, fp);
 while (s[0] != ' ' && !feof(fp)) {
 strcat(r, s);
 fgets(s, 2, fp);
 }
 memset(t, '\0', 20);
 if (!feof(fp))
 fgets(s, 2, fp);
 while (s[0] != ' ' && !feof(fp)) {
 strcat(t, s);
 fgets(s, 2, fp);
 }

 /* Add the link to the list. */

 k = (struct list *) malloc(sizeof(struct list));
 if (!k) {
 printf("error in memory allocation memfull");
 exit(0);
 }
 prev->right = k;
 strcpy(k->name, p);
 strcpy(k->filename, q);
 if (r[0] != '\0')
 sscanf(r, "%d", &k->type);
 else
 k->type = 0;
 if (t[0] != '\0')
 sscanf(t, "%d", &k->distance);
 else
 k->distance = 0;
 k->right = NULL;
 k->left = prev;
 prev = k;
 memset(p, '\0', 20);
 memset(q, '\0', 20);
 memset(r, '\0', 20);
 memset(t, '\0', 20);
 fgets(s, 2, fp);
}
fclose(fp);
}

```

```

/* This function can be used to print the information of the link.
 * This function is not used currently.
 */

void printlist(struct list *p)
{
 while (p != NULL) {
 printf("name %s filename %s type %d distance %d\n", p->name,
 p->filename, p->type, p->distance);
 p = p->right;
 }
}

/* This function is used to write the list of links into the file.
 * This function is used to update the link file.
 */

void writelist(struct list *p)
{
 char str[100];
 FILE * fp1;
 sprintf(str, "%sfff.c", lib_d);
 fp1 = fopen(str, "w+");
 while (p != NULL) {
 if (p->type) {
 if (p->right != NULL)
 fprintf(fp1, "%s %s %d %d ", p->name, p->filename,
 p->type, p->distance);
 else
 fprintf(fp1, "%s %s %d %d", p->name, p->filename,
 p->type, p->distance);
 }
 p = p->right;
 }
 fclose(fp1);
}

/* This function is used to free the list created above. */

void free_list(struct list *p)
{
 struct list *next;
 while (p != NULL) {
 next = p->right;
 free(p);
 p = next;
 }
}

/*
 * * * * *
 * Filename : Thesaurus.h
 * * * * *
 * Programmed by : N. Sunil Chakravarthy *
 * * * * *
 * Last updated on : 08.17.94
 * * * * *
 */

#include <stdio.h>

struct word_list{
 char word[20];
 struct word_list * right;
};

struct word_list *wordhead;
void print_list();
void freelist();

```

```

/*
 * * * * *
* Filename : Thesaurus.c *
* *
* Programmed by : N. Sunil Chakravarthy *
* *
* Last updated on : 08.17.94 *
* * * * *
*/

```

This file contains the function to look up a given word in the thesaurus file and load the equivalent words into a linked list, if found.

```

/*
include "thesaurus.h"
extern char lib_d[100];

int thesaurus(char word[80])
{
 FILE * fp;
 int success = 0;
 struct word_list *prev, *k;
 char s[1], p[82], str[100];
 char *q;
 int j = 0;
 sprintf(str, "%sthesaurus.dat", lib_d);
 fp = fopen(str, "r+");
 if (!fp) {

 /* Unable to open thesaurus file give error message. */

 printf("ERROR OPENING THESAURUS FILE ...\n");
 exit(0);
 }
 fgets(p, 80, fp);
 while (!feof(fp)) {
 q = strtok(p, "\n");
 strcpy(p, q);
 if (strcmp(word, p) == 0) {

 /* Word is found in the thesaurus file start
 * locating the equivalent words.
 */

 success = 1;
 wordhead = (struct word_list *)malloc(sizeof(struct word_list));
 wordhead->right = NULL;
 prev = wordhead;
 memset(p, '\0', 20);
 fgets(s, 2, fp);
 while (!feof(fp) && s[0] != '.') {
 while (s[0] != ' ' && !feof(fp)) {
 if (s[0] == '.')
 break;
 strcat(p, s);
 fgets(s, 2, fp);
 }

 /* One equivalent word is found, add to the linked list */

 k = (struct word_list *)malloc(sizeof(struct word_list));
 if (!k) {
 printf("MEMORY ALLOCATION FAILED..\n");
 exit(0);
 }
 k->right = NULL;
 prev->right = k;
 prev = k;
 strcpy(k->word, p);
 memset(p, '\0', 20);
 while (s[0] == ' ' && !feof(fp))
 fgets(s, 2, fp);
 }
 }
 fgets(p, 80, fp);
 }
}
*/

```



```
 fclose(fp);
 return success;
 }

/* This function can be used to print the equivalent words found in the
 * thesaurus, given a pointer to the word. This can be used after loading
 * the equivalent words.
 * This function is not currently used.
 */

void print_list(struct word_list *p)
{
 while (p != NULL) {
 printf("synonym %s\n", p->word);
 p = p->right;
 }
}

/* This function is used to deallocate the memory and destroy the linked list. */
void freelist(struct word_list *p)
{
 struct word_list *next;
 while (p != NULL) {
 next = p->right;
 free(p);
 p = next;
 }
}
```

## VITA

Sunil C. Nadella

Candidate for the Degree of

Master of Science

Thesis: A USER FRIENDLY INTERFACE FOR RCS AND ITS USE AS A SOFTWARE REPOSITORY

Major Field: Computer Science

### Biographical:

Personal Data: Born in Eluru, Andhra Pradesh, India, May 26, 1968, son of Venkataramana N. Nadella and Padma Nadella.

Education: Graduated high school from Kendriya Vidyalaya (C.L.R.I), Madras, India in May 1985; received Bachelor of Engineering Degree in Computer Science from Venkateswara College of Engineering, Madras University, Madras, India in May 1990; completed the requirements for the Master of Science degree in Computer Science at the Computer Science Department at Oklahoma State University in May 1995.

Experience: Employed by Oklahoma State University, Agronomy Department, as a programmer from 1993 to September 1994.