Rochester Institute of Technology
RIT Digital Institutional Repository

Theses

10-1-1986

TCS: a version control system

Rosita Caridi Scheible

Follow this and additional works at: https://repository.rit.edu/theses

Recommended Citation
Scheible, Rosita Caridi, "TCS: a version control system" (1986). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

TCS
A VERSION CONTROL SYSTEM

by

Rosita Caridi Scheible

Submitted in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Approvals: Peter H. Lutz

Peter H. Lutz (chairman)

Michael J. Lutz

Michael J. Lutz

Will J. Stratton

William J. Stratton

GRADUATE COMPUTER SCIENCE DEPARTMENT
ROCHESTER INSTITUTE OF TECHNOLOGY
Rochester, NY 14623

October, 1986

ed sample statement for granting or denying permission to
duce an RIT thesis.

of Thesis TCS A Version Control System

Rosita Scheible Rosita Scheible hereby (grant,
permission to the Wallace Memorial Library, of R.I.T., to
duce my thesis in whole or in part. Any reproduction will
e for commercial use or profit.

Or

_____ prefer to be
cted each time a request for reproduction is made. I can be
ed at the following address. _____

_____

Nov. 21, 1986

CONTENTS

CONTENTS

CHAPTER 1

INTRODUCTION TO TCS AND VERSION CONTROL

1.1 INTRODUCTION

TCS (Text Control System) is a version control system implemented under UNIX(tm). It records changes made to a text (ASCII) file, providing facilities for the storage and retrieval of multiple versions of a file, in such a way that no version, once stored, is ever lost. TCS provides basic controls over how changes are stored, keeps conflicting changes separate, and provides a convenient mechanism for identifying any version. In addition, for each version, TCS keeps a record of who created the version, when the version was created, and any user submitted documentation text.

The purpose of this thesis project is to demonstrate and implement a version control system that reflects many of the positive features found in such systems on the software market today.

Several of these systems will be discussed in subsequent sections of this paper. The intent of this project is not to implement a system suitable for a production environment, but TCS does, indeed, have many immediately useful features. It provides a simple, yet functional, user interface consisting of a handful of commands that the user can master quickly. Even in its present form, it is suitable for use by an individual or small project team consisting of a few individuals.

1.2 THE NEED FOR VERSION CONTROL SYSTEMS

Most commercially available version control systems arose from the need of software manufacturers to control their own engineering environments.

During the software development and maintenance cycle, the components of a software system (source code, object code, documentation, input file, etc.) will undergo several changes. Several versions of selected components begin to evolve and this in turn results in multiple versions of the software system itself. More often than not, multiple versions must be permanently stored for immediate retrieval by various groups

within the software project. Furthermore, it is often difficult to predict which version will be needed and for what purpose at any given time. It may be important to store all versions.

Identifying the versions themselves, who they belong to, when and for what purpose they were evolved became central issues throughout the software life cycle.

A version control system insures that no version (or any important information related to the version) is ever lost, once stored. It provides a means for control over changes. The most crucial control is the system's requirement that two or more individuals do not try to make a change to the same version at the same time, which would result in incompatible changes.

Once a change is stored, a version control system records who made the change and requires the user to enter a comment regarding the purpose and nature of the change.

Given these simple controls, any individual within the project should be able to make changes to a component without fear that the work will be inadvertently corrupted. Furthermore, if two

individuals must work on a version concurrently, the ability to identify who is working on the version gives them the chance to either coordinate their efforts or agree to keep changes separate. Should some problem arise with any version (a bug in the code, for instance), the person responsible for the change is readily identified. This person is most familiar with the change and would consequently be the central figure responsible for resolving the problem. In large engineering environments, more elaborate controls are often implemented. Control over changes may mean that only chosen individuals be allowed to make changes to any particular version, or that strict naming conventions be instituted. Naming conventions provide a simple, yet powerful, vehicle for associating changes with a specific project or version of the entire software system (i.e., a system configuration).

Software is not the only entity that requires version control. Large documents, like user manuals, reference guides, and internal software documentation are continuously under revision. Like software, these are stored on-line, and require control over their changes. A version control system can offer controls for these as

well.

Whenever multiple versions of an entity must be stored for immediate on-line retrieval, the question of available disk space arises. While disk space is continuously coming down in price, it still remains an expensive proposition to store redundant data in large bodies of information. To conserve disk space, a version control system stores only the changes (or "deltas") to any version. That is, it stores the differences among versions rather than all versions in their entirety.

The rationale behind version control within software development environments is applicable to any environment where multiple versions of on-line files must be stored and are continuously undergoing revision. Version control systems are used to store government and legal documents, hardware configuration information, VLSI and PCB layouts, etc.

1.3 BASIC REQUIREMENTS

With the need for version control in mind, we can now state the minimum requirements of such a system. The system should:

1. Store multiple versions of a text file in a space efficient manner.

2. Protect the integrity of each version. (Once stored, the version is always readily available. It always looks the same regardless of any changes which evolve subsequent versions.)

3. Permit concurrent development of any version, but insure that incompatible changes are disallowed.

4. Allow any version to be modified at any time and allow any subsequent version to be stored.

5. Provide a mechanism for the identification of any version, insuring that the user has control over version identification.

6. Record, for each version, who created the version and when it was created.

7. Allow the user to enter additional descriptive information about the version at the time of its creation.

8. Make information on any version readily available to the user .

1.4 THE TCS SYSTEM

The TCS (Text Control System) stores and retrieves multiple versions of an ASCII file. To conserve space. TCS stores only the original version in its entirety with subsequent versions stored as deltas of the previous version. Thus, only the differences among versions are physically stored. When the user requests a version, TCS builds the version by applying deltas to previous versions and delivers the requested version to the user.

The user views the collection of versions as forming arbitrary branches in a tree. Each branch is composed of a chain of nodes representing the deltas for each version. Deltas are inserted at the end of a chain, never between deltas. This insures that a version looks the same at all times. When a version along a chain is requested, TCS applies all deltas beginning with those for the first version in the chain up to and including those for the requested version.

A version is identified by a tree name and a version number within the tree. The version number is of the form: r.d where r is a release level, defined by the user and d is the delta within the release.

When the user creates a TCS module, the original version is stored as: tree 0 version 1.1. Tree 0 is the main tree of any module from which other trees can be defined. As versions are added to a tree, TCS automatically numbers the new version by adding 1 to the previous version's delta number. The user can override this by requesting that the new version have the next release number. In this case, TCS adds 1 to the previous version's release number and makes the new version's delta number 1.

Any node in any delta chain can be split into one or more trees, thus allowing paths of diversion along the delta chain. The user chooses the tree name with the exception of the main tree's name which is always "0". TCS retrieves a version within a tree by applying all deltas for the tree's parent node and deltas within the tree's delta chain. In a sense, each tree can be viewed as a separate module having its own chain of deltas, its

original version being that of the parent node.

Figure 1-a below illustrates the tree structure of a collection of versions of a file called "program.c". Tree names precede their delta chains. Each delta is shown in the form of r.d with -- marks separating the deltas. Each delta was created from the versions represented by deltas to the left of it along a chain. The main tree, 0, has 4 releases, with the latest version being 4.1. Trees: "treel," "treel_a," "tree_a," "tree_b" each have their own delta chains. Deltas for these trees are applied only when the user calls for a version within the tree. For instance, if the user calls for version "treel_a 1.1" the deltas 0 1.1, 0 1.2, treel 1.1, treel 1.2, and treel_a 1.1 are applied and all other deltas are ignored. If the user called for version "0 2.1," then deltas 0 - 1.1--1.2--1.3--2.1 are applied and all others are ignored.

Any node can have any number of subtrees or "children" trees. The depth of any tree is limited only by certain "hard" limits, on the total number of trees, imposed by the system. That is, all trees are created equally. The only case in which TCS will not create a new tree at the user's

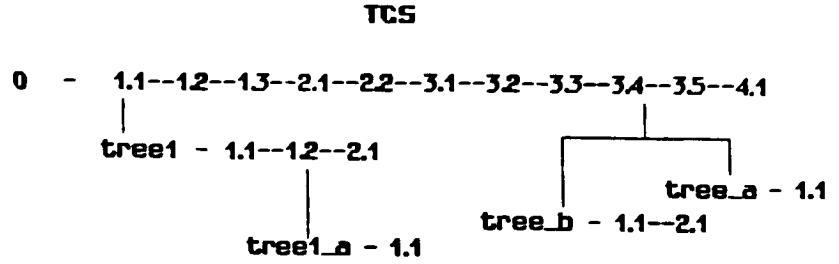


Figure 1-a

request is when its tree table is full. The tree table can currently accommodate 100 trees. There can be any number of deltas within a release.

The concept of a tree as a path of divergence within a delta chain is an integral part of version control. There are many times when it is necessary to insert a delta of a version in the middle of a delta chain. If TCS were to permit this, the integrity of subsequent versions within the chain would be corrupted since versions are built from previous versions along the chain. By creating a tree of the intermediate node, the user is now free to change this version as he/she chooses. Thus, any version within the module can be changed and its changes stored at anytime, without affecting previous or future versions of other nodes.

Trees also allow for concurrent changes to a version. In order to make a change to the last version in a delta chain, the user must reserve the version for change purposes. Once reserved, any attempt by another user to add a delta to that chain is shut out until the user who requested the reservation either cancels the reservation or adds the new version to the chain. This control protects the integrity of the new version. However, two or more users can concurrently store their changes to a version by creating a tree of the node in question and reserving their respective trees' last version for change. Thus changes made by two or more users can be stored separately without affecting each other.

Information regarding the module file is always readily available to any user. TCS provides the SHOW command for tracking the structure and state of the delta tree for any module. SHOW displays a tree's path name, will tell whether or not a tree is reserved for change and by whom, and show the last revision date for a tree. It will also show the release chain for any tree and the highest delta in each release. To help track concurrent development, the SHOW CHILD facility will display the children trees, if any exist, for

any node. SHOW CHILD also tells whether the child tree(s) is reserved for change. If the tree is reserved, SHOW displays the reserver's ID, and the time and date on which the reservation was made.

CHAPTER 2

A SURVEY OF VERSION CONTROL SYSTEMS

This chapter will consider several commercially available Version Control Systems:

SCCS (under UNIX)

DEC's CMS (under VAX/VMS)

RCS (under UNIX)

Appollo's DSEE

Softool's CCC

SMDS, INC.'s Aide-de-Camp.

In much of its internal and external structure, TCS resembles SCCS, UNIX's Source Code Control System. Both allow the user to view the collection of versions as nodes in a tree, operate on a single "Module" at a time, and incorporate the "release" and "delta within the release" idea. TCS's version storage mechanism, the interleaved file, and accessing algorithms are based on those for SCCS described in Rochkind's article "The Source Code Control System" [14]. The storage and

accessing algorithms are explained in a later chapter . SCCS AND TCS use the UNIX "diff" program to isolate differences.

Of course, SCCS differs from TCS in many respects. SCCS makes no provisions for assigning symbolic names to any node, while TCS provides the tree name capability. Note that if a TCS node is treated as an empty tree, then any node can be assigned a symbolic name. TCS also encourages extended change documentation by accepting a comment of up to 1000 characters on a new version. Since comments are variable length and not accessed as frequently as the versions themselves, they are kept in a separate history file. This saves the "GET" program the time and trouble of reading through or around all comments to get to the body of the TCS file.

Rochkind's description of SCCS is significant in that it provides a model for Version Control. All version control systems store deltas rather than entire versions. All, except the RCS system, manage the deltas through the use of the interleaved file, which stores all versions in one file. Deltas are interspersed throughout with control records surrounding each delta to identify

the version that created the delta. All of the above systems, at the very least, provide for concurrent development, record when a change was made and by whom, require that only one person at a time check out a version for change purposes, and require a descriptive comment when a new delta is installed in the file. In addition, they provide a mechanism like the "SHOW" command that display information on the structure of the revision tree and about each version. In short, all meet the minimum requirements outlined earlier.

TCS, SCCS, CMS, and DSEE are similar in their user view. Revisions appear as incremental nodes in delta chains with branches from any node handling paths of divergence. These operate primarily on single files.

CCC and Aide-de-camp differ dramatically from the other version control sytems in their user view. These handle collections of files in a centralized data base, providing full configuration management for an entire system of data.

In general, version control systems differ primarily in the extent to which protection mechanisms are employed, in their facilities to handle entire configurations, in their the ability

to automatically merge divergent versions, and in their the facilities to handle additional information about a version besides a descriptive comment at creation time.

2.1 SCCS (SOURCE CODE CONTROL SYSTEM)

Since the original publication of SCCS [14] in the early 1970's, the system has undergone several revisions. The following discussion is taken primarily from the UNIX System V Users Manual [8]. A critique of SCCS (in comparison to RCS) is found in [11].

SCCS operates on a "module" at a time. A module refers to a convenient unit of source code usually a subroutine or macro [14] that would normally be stored in a single ASCII file. All information about the module is stored in one UNIX file. Tables, protection information and user submitted comments form the "header". The body consists of the original version, deltas, and control records used to recreate the versions.

Like TCS, versions develop along a main chain of deltas and are grouped into releases. When a new delta is added to the chain (Delta operation), SCCS automatically numbers it with the next highest

delta number within the release.

An "administrator" over the module defines the next release. SCCS also allows the administrator to specify the initial release number and whether or not release numbers are skipped in the release sequence. This serves a valuable function since in a typical engineering environment, all versions are not kept on line for all time. Once a software product is released to the field, it is that version and a back version that are of primary importance. Obsolete versions are stored on tape. And new SCCS modules are created from the most current ones. By allowing the user to number the initial release, the sequential numbering system of releases is preserved. Note that in TCS and RCS the initial release number is always 1.

To allow for concurrent development, the administrator can create branches emanating from any node in the module tree. SCCS automatically numbers a new branch with the next higher branch number for its parent node. Several versions can develop along any branch, and SCCS will sequentially number them, with a "sequence" number. So a version is uniquely identified by R.L.B.S. where R is the release number; L is the delta

within the release; B is the branch number; and S is the sequence number within the branch. Figure 1-b illustrates an SCCS revision (delta) tree. The node denoted by "Node X" is identified as version 2.1.1.2. Node Y's identification string is 2.1.2.1.

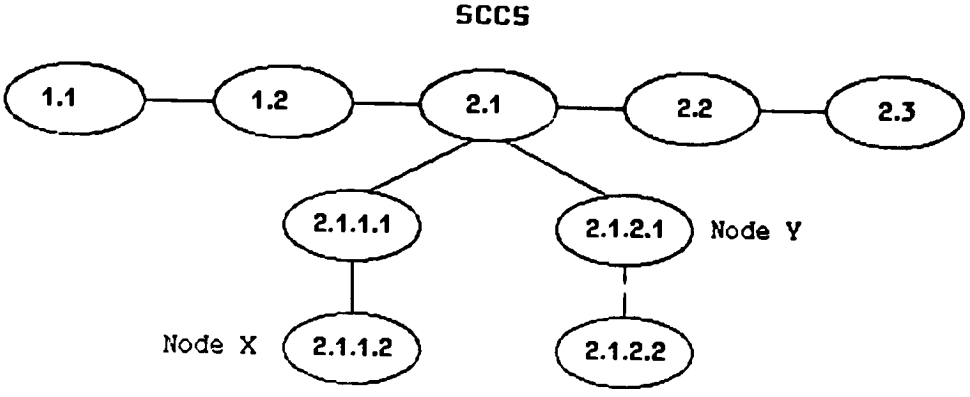


Figure 1-B

When retrieving a version (GET operation), for read or change purposes, the user can specify that one or more deltas be included or excluded. The inclusion mechanism forces deltas along separate branches to be applied, causing automatic merging of changes. The "merged" version is delivered to the user, who can then store this version as the next one in a delta chain. The exclusion mechanism instructs the system to ignore

deltas that would normally be applied for the version. Excluding deltas is handy for correcting mistakes in a stored version.

When a version is retrieved for editing purposes, the user who made the reservation is the owner of a lock on the version. Before someone else can add a version, this user must either relinquish the lock or add a new version. A module administrator can break the lock to allow another user to add to the delta chain, or create a branch for concurrent work. Thus, work on the module can proceed in the absence of the person who made the original reservation. It is important to note that the administrator breaks the lock. This can serve to insure that the parties involved are in communication regarding the state of the module and that deltas are not created from the wrong versions.

An administrator can enforce control over who makes changes within a release. SCCS keeps a list of users who can add deltas to a release. If the list is empty, then anyone with write access to the SCCS module file can add deltas. The administrator controls additions and deletions from the list. Furthermore, releases can be locked so that no

further changes can be made to any node in the release.

Additional highly useful features include retrieval by cut off date where the system gets a version ignoring deltas made after the cut off date. An administrator can specify that the system prompt the user for a Modification Request Number each time a delta is created. This helps to justify the reason for creating deltas and track its effects. SCCS will also compare two or more versions within a module (SCCS diff command).

2.2 CMS (CODE MANAGEMENT SYSTEM) [4]

Digital Equipment Corporation's CMS runs on the VAX under VMS. CMS manages libraries of related files. Each file in the library is known as a CMS ELEMENT. To facilitate element identification, the user can subdivide Libraries into "GROUPS" of elements.

Within each element, Versions develop along a main line of descent. As deltas are added, the system sequentially numbers the nodes. Paths of divergence from the main line are given by a variant letter. A variant letter is user specified. Any node, even variant nodes, can have

variants, so the delta tree can reach any depth. Furthermore, any node can be placed into a "CLASS," giving the node a symbolic name. Nodes from several elements can belong to one class. Figure 1-c shows a delta tree having a node whose "CLASS" name is "baseline." (GENERATION 4 of the main line of descent). BASELINE has a variant branch A with 2 deltas A1 and A2. Node BASELINE/A2 has a variant A also with deltas A1, A2, and A3. The node with the class name "proj002" is version 4A2A2 or BASELINE/A2/A2.

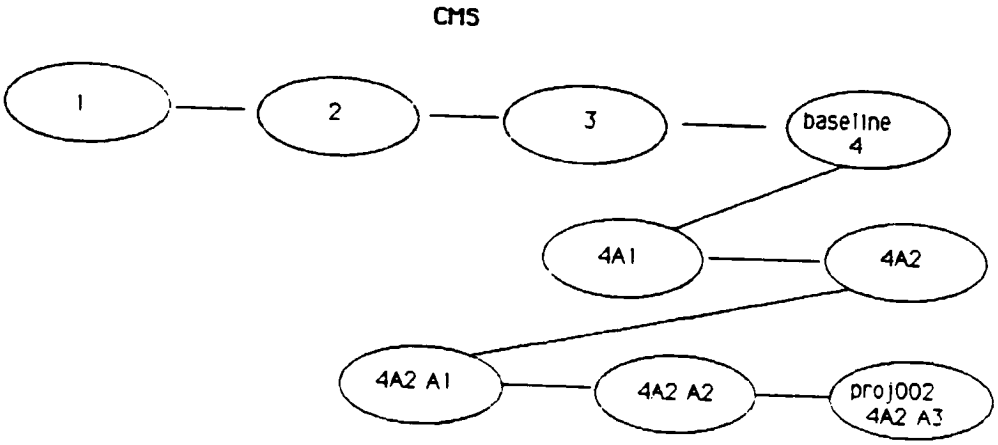


Figure 1-C

In order to add a delta to any branch, a user must "RESERVE" the last node in the branch. If another user attempts to Reserve the same node, CMS informs him of the previous reservation. His

options are to either quit or proceed with a concurrent reservation. Several users can reserve a node at a time. However, the first to "REPLACE" the version creates the next generation. When the next person attempts to replace the modified version, the system informs him of the previous replacement. His options, in this case, are to create a variant node or merge his changes with the latest version by reserving the last node using the /MERGE option.

When the user retrieves a version using the /MERGE qualifier, CMS automatically merges the versions and delivers the merged version to the user. Line conflicts are flagged in the merged file. (A line conflict occurs when the two versions changed the same line of text.) The user reviews the file, resolves any conflicts and then issues a REPLACE to permanently store the changes. The /MERGE qualifier can also be used on a "FETCH" (read only retrieval). So, it is not necessary to reserve an element before seeing the results of a merger.

CMS relies on the VMS file protection mechanism for defining who can make changes to any element within the library. A CMS library is

actually a VMS directory defined by the user. Anyone with write access to a file on this directory can add deltas to any version within the CMS ELEMENT (which is actually a VMS file). Versions cannot be selectively write protected as under SCCS.

2.3 DSEE (DOMAIN SOFTWARE ENGINEERING ENVIRONMENT) [5,7]

DSEE is a collection of software engineering tools for the APOLLO micro computer systems. DSEE runs under the APOLLO AEGIS operating system. Version control is a major tool within this engineering environment.

Like CMS, DSEE groups user files into libraries. Each file is a library "element." Protections are established at the library level, not at the file or version level. Anyone with "MEMBER" privileges to the library can modify an element.

In DSEE, the version tree evolves in a manner similar to CMS. Generations of nodes are sequentially numbered starting with generation 1 along any branch. Any node can be given one or more symbolic names. The tree can reach any depth.

Variants from a line of descent are handled in a slightly different way. If a line of descent is reserved for editing, any other attempts to reserve it are shut out. If a second user needs to modify the same version, he creates a branch, giving the branch a symbolic name. A member can also render a line of descent "obsolete," meaning that it cannot be extended but anyone with read privileges can still read any version along the branch.

To further facilitate concurrent development and cooperation among users, DSEE provides a MONITOR feature. When a user places a monitor on an element, this user is notified of any modifications to the element. A person issuing a "RESERVE" is also informed of the monitor.

DSEE provides a means of merging separate lines of descent. The merge operation is performed interactively, so that the user can see and control the results immediately. DSEE displays lines from each version. The system cannot merge lines without the user's permission.

To conserve space, as with other systems, DSEE is delta based. However, it uses "blank compression," for suppressing leading blanks on all lines stored. As with other delta based systems

which manage versions via the interleaved file, the time to retrieve a version is directly proportional to the size of the file. All versions are retrieved in equal time. DSEE assumes that the latest version will be retrieved most often. To save retrieval time for this version, it keeps a cache of the latest version of all elements in the library. That is, this version is stored in its entirety outside of the interleaved file. This, of course, increases the use of disk space, but other space saving measures, like blank suppression, help offset the effect.

An additional significant feature under DSEE, is a Configuration Management facility. The user can define components of a system through "configuration threads." These threads are lists of user specified versions of each element composing a system configuration. This, coupled with the ability to give nodes a symbolic name (the nodes belonging to a specific configuration within each element can be given the configuration name) and a wildcarding capability which allows the user to specify several versions having the same name provide a powerful configuration management tool.

2.4 RCS (REVISION CONTROL SYSTEM) [1,11]

To the user, RCS (which is implemented under UNIX) appears to treat versions in the same way as SCCS and TCS. Like these, RCS groups revisions into releases along the main trunk of the revision tree (TCS allows releases along any branch). As in SCCS, RCS numbers branches. Branch nodes, themselves, can have branches. As in TCS, the user can assign a symbolic name to any branch as well as to individual revisions.

RCS is unique in that it does not use the interleaved file to store deltas. Rather, deltas are stored as a series of edit commands known as "Reverse" deltas upon the latest version (of the main branch). Branches are stored as foreward deltas. That is, versions along branches are constructed by applying a series of edit commands to the previous version.

Figure 1-d shows an RCS revision tree whose main branch consists of versions 1.1, 1.2, and 2.1. The arrows show the delta direction. RCS stores the entire version 2.1, the edit commands to transform 2.1 to 1.2, and the commands to transform 1.2 into version 1.1. Node 1.2 has two branhes: one named "temp_fix" (version 1.2.2.1) and one

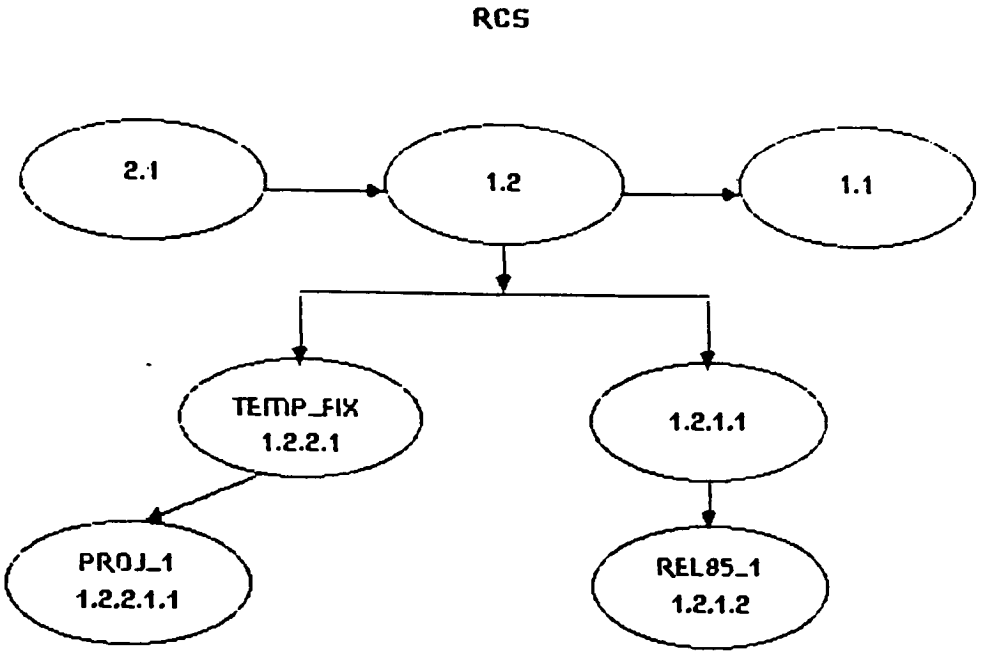


Figure 1-d

identified by the system generated number "1.2.1.1". The branch node "temp_fix" also has a branch identified as "proj_1," or by its version number, "1.2.2.1.1." Any branch node is shown as a foreward delta. Along any branch, a series of edit commands on the previous version is applied in order to generate a version.

Reverse deltas provide an advantage in access time for the latest version, the assumption being that these are accessed the most. The disadvantage would follow that generations other than the latest

takes time proportional to the number of deltas applied.

As in the other systems, RCS requires a version to be reserved or "locked" for editing before the user can add a delta. However, RCS provides a flexible command for breaking a lock so that work can proceed uninterrupted in the absence of the lock owner. SCCS also provides this lock breaking, but it must be done by an administrator with special privileges. No such privileges are required under RCS. But, unlike in SCCS, RCS notifies a lock owner of a "break in" via a mail message containing the breaker's identification and a commentary. To allow any user this privilege does not seem to cause excessive conflict in practice. According to Tichey [1], "the automatic mail message attaches a high enough stigma to lock breaking, so that programmers break locks only in real emergencies, or when a co-worker resigns and leaves locked revisions behind."

RCS facilitates Configuration Management by allowing symbolic names on any node and a wildcarding capability that causes retrieval of several files at a time. For example, suppose that several RCS files with a .C extension existed on an

RCS directory and that each file contained a version called "reL85_1." The RCS "CHECK OUT" command, CO -rrel85_1 .C, would retrieve the "reL85_1" version from each .C file. In addition, RCS allows retrieval by cut-off date (as does SCCS).

RCS provides a convenient MERGE command that functions in a manner similar to the CMS /Merge facility. In addition, RCS's "CHECK IN" (equivalent to TCS's PUT) operation permits the use of a pathname on both the RCS revision file and the user's working file which contains the new verison. This convenience saves the user from the nuisance of copying the working file into a file with the same name as the revision file on the revision file's directory. Neither SCCS nor TCS provide this facility, as yet.

RCS keeps a list of users who are allowed to update RCS files. Thus, a system administrator can set protections at the file level. However, RCS does not have a provision for selectively protecting specific versions.

2.5 CCC (CHANGE AND CONFIGURATION CONTROL) [2,3]

Softool Corporation's CCC is a powerful

configuraiton management tool, running on several machines under several operating systems (among them are: VAX/VMS, VAX/UNIX, IBM/CMS, IBM/MVS, IBM PC/MS_DOS, APOLLO/AEGIS, DG/MV_UNIX). CCC controls and tracks changes to binary as well as to ASCII and EBCDIC files.

The user sees a hierarchical organization of information in CCC. The hierarchical levels, termed DATA BASE, SYSTEM, CONFIGURATION, and MODULE, are illustrated in Figure 1-e.

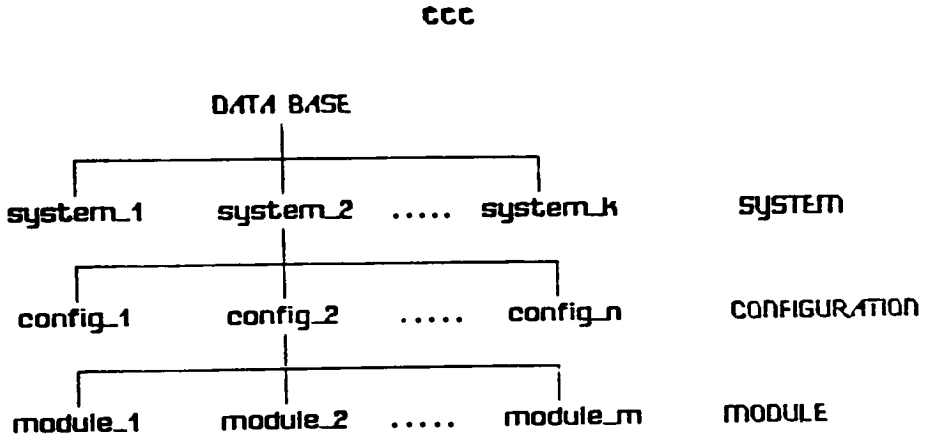


FIGURE 1-e

The data base is composed of "systems" of data. The configuration level represents the versions of any given SYSTEM. The user can choose to further divide configurations into MODULES representing the components of the configuration.

Physical data entities stored under CCC are termed "TEXT data structures" and can be stored at any level within the hierarchy. The user can group related text data structures by defining text TYPES. A version of text is identified by a system generated version number or by a user supplied CHANGE NAME. Version control is provided on any TEXT data structure at any level within the hierarchy. The user has the option of storing changes either as a delta of the last version of the TEXT data structure or as a separate version in its entirety.

CCC provides the usual locking mechanism for a change to a TEXT file. To effect concurrent development, the user creates a new system configuration or module data structure and inserts the TEXT structure into the new structure. Of all the systems mentioned, CCC provides the most powerful and elaborate protection mechanisms. CCC keeps a list of all users, which data structures they can access, and passwords to data structures within the hierarchy. In order to access any information, the user logs into a specific system or configuration and, in doing so, must supply his/her password to the structure.

The "cast of characters" within a CCC environment are a "Data Base Administrator," "Managers," and "Users." The Data Base Administrator has free reign over the entire data base. This person initializes and maintains the data base, has read/write access (MANAGER privileges) to any data structure at any level of hierarchy, and defines the MANAGERS and USERS. MANAGERS have special privileges to the system or configuration. Managers delegate read/write privileges to USERS within their domain. They determine who can make changes to a TEXT data structure and who can physcially delete structures. The "USER" can insert new TEXT structures, make changes, and logically delete data structures. Under CCC, the organization can create an environment that mimics the organization's chain of authority. In addition to these protections, CCC provides an encryption facility, in which TEXT structures can be stored in up to 9 levels of encryption.

So that the organization can tailor its use of CCC to its individual environment, CCC has a powerful MACRO capability and an internal editor which can be used to make highly controlled automatic changes to TEXT structures. Furthermore,

the internal editor gives the user the option of making changes right within the CCC environment. Contrast this with the other systems where changes are made using the operating system's editor exclusively. Under CCC, TEXT files, of course, can also be "EXPORTED" (read from the data base), edited within the user's environment, and the new version "IMPORTED" back into the data base. As with RCS, the user can specify a name for the imported file other than the data base text file name.

When a user enters a new version into the data base, CCC requires a descriptive comment of the change. Descriptions entered under SCCS, RCS, CMS, and DSEE are limited to one line. "One liners" are sometimes inadequate to describe the nature of a change. Under CCC, the user enters a multi-line description (0 to 436 characters). TCS also allows multi-line descriptions. The idea for line prompting under TCS was taken directly from CCC. CCC prompts with the number of characters entered on preceding lines.

2.6 AIDE-DE-CAMP [10]

The Aide-De-Camp software engineering

environment, from Software Maintenance and Development Systems, Inc., is implemented under the UNIX, and VAX/VMS operating systems. The system manages configurations within a centralized data base. Aide-de-Camp captures incremental changes to individual files through the use of "CSETS" (change sets). CSETS are composed of related files undergoing revision by an individual or project team. In practice, organizations often group related changes into "projects." By tagging a CSET with a project identification code, the user can easily track all projects. When the user creates a CSET, Aide-De-Camp requires that the user enter a remark describing the reason for the CSET. This remark can serve to further facilitate project tracking.

Aide-De-Camp provides a simple yet elegant user view of changes to configurations. A version of the system is defined via a "list equation" which gives the relationship between an already installed version and one or more CSETS. Suppose a version (configuration) V exists and change sets A and B contain changes to a file in V and/or new files to be combined in V, a new version V1 is defined by the equation:

$$V1 = V + A + B.$$

In order to delete changes from a version, the "-" operator is used. The equation:

V2 = V1 - B

would delete CSET B's changes in the new version V2.

The system supports concurrent development by allowing CSETS to have files in common. The user will place "conflicting" changes into separate CSETS and later merge the changes into a new version. Consider a version V consisting of files F1, F2, F3, F4, F5. CSET A consists of changes made to files F1 and F2, and a new file F6. CSET B changes files F1 and F3. If V1 = V + A + B, then V1 consists of:

- those files in V not changed by A or B
- the version of F2 from A
- the version of F3 from B
- the line by line merger of versions of F1 from both A and B
- the new file F6 created by A.

Like CCC and TCS, Aide-De-Camp will store an extended commentary on the nature of each change. Whenever a change is installed within a CSET, the system requires the user to submit an "abstract" describing the change. The abstract is a file containing the documentation. Thus, the system encourages full documentation of changes.

Primarily intended for software engineering and the source code storage, Aide-De-Camp records relationships among entities in the data base. These relationships help the user to identify actions to be taken when object files are generated from changed source. The user can assert that file GT "includes" file RX and/or define object libraries and assert the procedures (subroutines) contained in the library. The system also stores information such as, "file X 'contains' procedure Y" and "Y calls procedure Z." Such information helps to identify calling trees and which files are affected by a bug fix or enhancement. It also helps to avoid duplication of subroutine names when a new file is added to a software system. Relationships are stored in relationship tables. The user can always read or make changes to these tables. The system records changes to these tables in CSETS in the same way that changes to source are

recorded. So a version can also include changes to these tables.

Protections are set at the data base level. Any user with write privileges to the data base can make changes to any file, create CSETS, assert and change relationships, etc. In other words, the user cannot selectively protect individual entities.

2.7 CONCLUSION

While version control systems (TCS included) differ from each other in how they define their user view of how information on one or more files, within their domain, is stored all provide the basic functionality needed for version control. They provide an automatic way of logging who made a change and when. For instance, TCS stores the user's login ID and date from the operating system. Each system stores incremental changes but also provides for paths of diversion. Concurrent changes to the same version are kept separately. To conserve space, all systems are delta based. With the exception of the RCS system, these systems store and retrieve deltas using the interleaved file and an accessing algorithm similar to the one

described later in this paper. All store information on the nature of a change in the form of a user commentary submitted when a new version is stored. While the system cannot provide automatic control over how informative this commentary is, it does encourage the documentation. The important elements of recording who, when, why, and how a change was made are present in any Version Control System.

Usually version identification is implemented via an automatic sequential numbering system like that in TCS. In addition, most systems allow the option of symbolic names for version identification. A shortcoming of SCCS is the absence of this feature. Without it, the user is forced to remember the meaning of each version number. SCCS, RCS, and TCS support release groupings of versions whereby specific incremental changes can be related to milestones within a project. By supplying the release number alone on a "get" type operation, the user can retrieve the highest delta within the release. The user need not remember all version numbers and their meanings. In addition, TCS allows releases to develop along any branch in the version tree, so that any node can be given a "milestone" name. A

branch is then viewed as a separate module with its own main chain of deltas.

TCS, CCC, and Aide-De-Camp encourage full documentation of changes by providing an extended comment capability when a delta is added. The system assumes the task of associating documentation text with its corresponding version. Other systems associate a one line comment with each version. In all systems, the quality of this documentation (that is, how well it describes the nature of a change) is the responsibility of the user.

TCS, like most systems, relies on the operating system's protections. Anyone with read/write access priviliages to a TCS file can add deltas anywhere within the version tree. SCCS and CCC provide their own protections whereby protections are set on selected versions within the revision file. In large engineering environments, the extended protections may be mandatory requirements of the system.

Without exception, all systems "lock" versions undergoing change. This ensures that changes are incremental; that is, each delta is created from the version that the user intends. Locking

versions, however, can hold up progress on a project, if the version is locked for an excessive length of time or in the absence of the lock owner. CMS gets around this problem by permitting multiple reservations on a version, under controlled conditions. SCCS, RCS, and TCS provide a means for breaking a lock. TCS provides the CANCELC command, but this command must be issued by the owner of the lock. Making lock breaking a conscious effort on the part of the lock owner insures that this person is aware of the who, why, and what changes are being made to a version that he intends to change. For extreme circumstances when the lock owner is not available, TCS provides the XCANCEL command which breaks a lock without asking any questions about the breaker's ID. This command is intended for use by some super-user who would take responsibility for informing the owner of the "break-in."

A significant feature present in most systems is an automatic merging facility. TCS, as yet, does not have this built-in feature. The merging issue warrants a stern caution. The user must always keep in mind this is strictly a clerical function of blindly merging lines. There is no intelligence here. The system has no way of

knowing whether the merged version contains the functionality that the user needs. For this reason, no system will immediately store a merged version. A merged version is always delivered to the user for inspection and editing if need be. DSEE goes a step further by requiring the user to inspect the version as it is built. While merging provides a convenience for the user, there is always the danger that the user, forgetting that automatic propagation of separately stored changes is purely clerical, will immediately store the merged version. In this case, the user is assuming that the system has determined that the functionality of the separate versions are compatible.

The author asked several software engineers to comment on merging. These engineers work on a large software system consisting of approximately 8000 files of source code. Conflicting changes is a central issue in this environment. Their responses to the question, "Is automatic merging a desirable feature for a code control system?" ran mostly along the lines of: "I wouldn't trust it;" "It would encourage sloppy programming;" "I'd rather recode than unscramble merged files."

Under close scrutiny, automatic merging is nice, but usually not necessary. The authors of SCCS saw fit not to make automatic merging easy for the user. The early verisons of the CCC system do not provide this ability. It was installed into CCC after its 2.0 release. While TCS does not provide for automatic mergers, the accessing algorithms are implemented in such a way that this feature could be added later.

Several of the systems discussed, provide other desirable features which could later be incorporated into TCS. Among these features are:

- retrieval by cut off date
- an exclusion mechanism whereby the system facilitates mistake correction by skipping deltas that the user specifies.
- the ability to specify path names for the user file containing a new version to be added to a module
- the ability for the user to selectively lock releases and trees from further changes

- the ability for an administrator to selectively read/write protect specified versions, releases, or trees

- a built-in wildcarding capability that facilitates the retrieval of related versions from several files

TCS contains the absolutely necessary features for version control. That is, it meets the minimum requirements outlined earlier. However, most important, the TCS programs store the information necessary for the enhancements listed above. The programs are structured in such a way that these enhancements can be added later with relative ease.

CHAPTER 3

TCS COMMANDS

This chapter contains two major sections: TCS USER COMMANDS and TCS SUPER-USER FUNCTIONS. The first of these sections contains details for each TCS command that the general user issues. Each TCS user command is presented in alphabetical order, on a separate page, so that the user can easily reference any command.

The last section, SUPER USER FUNCTIONS, outlines those commands issued by some administrator or the TCS programmer. Details for these are found in the command files (shell scripts) in Appendix F.

3.1 TCS USER COMMANDS

GET	Retrieve a version for read only or change purposes.
PUT	Store a new version along with information about the version.
CREATMOD	Create a TCS module file from the original version.
CREATREE	Create a child tree of a node in the TCS module.
SHOW TREE	Show (display) informatin about a tree or all trees.

SHOW CHILD	Show (display) the children of a node in the module.
CANCELC	Cancel the change reservation on a tree.

3.1.1 GENERAL FORM OF A USER COMMAND

The syntax of a user command is similar to UNIX commands with the exception that required arguments are submitted first, followed by optional arguments. In general, a command is of the form:
command <arg1> <arg2>...<argN> [-opt1 -opt2 ... -optN]

Required arguments are shown within < > marks. These directly follow the command, are positional arguments, and are separated by white space (blank or tab).

Options are shown within [ ]. Note: the user does not type the [ ] marks on the command line.

- An option is given by an option letter preceded by a "-".
- These can be submitted in any order.
- Options requiring an argument are shown as: -l <arg> where l is the option letter and <arg> is the required argument.

. Option arguments must directly follow the option letter. Any string of characters following the option letter is assumed to be the required argument.

GET currently requires that its options be followed by white space. See details of the GET command.

SHOW allows for optional white space separating its option letters and any option arguments. Options NOT requiring an argument may be catenated into an option string preceded by a "-". See the SHOW command for details.

CANCELC

Cancel the change reservation for a tree. CANCELC checks the user's login ID to ensure that it matches the user ID on the change reservation for the tree. If the ID's do not match, the reservation is not cancelled. Once a reservation is cancelled, other users may execute the GET operation with the -c option. See GET.

cancelc <module name> <tree name>

<module name>	name of an existing TCS module.
<tree name>	tree name whose reservation is to be cancelled.

CREATMOD

Create a TCS module from the file given by the <module name> argument. CREATMOD looks for a file called <module name> in the current working directory and stores the entire contents of the file as version "0 1.1" in the new "<module name>.t" file. The TCS module must not already exist in the current working directory.

The user is prompted for description information for this initial version as on a PUT operation. See details on PUT. CREATMOD creates a history file for the new module and enters the user's login ID, time, date and description information into the history entry for version 0 1.1.

creatmod <module name>

<module name>	name of the ASCII file on the current working directory. <module name> can by any valid UNIX file name. To avoid confusion with files created by TCS, <module name> should NOT begin with "t." or "h." or end with ".t". Path names are NOT permitted.

CREATREE

Create a child tree for a node in a TCS module. An empty tree is entered into the tree table. The new tree's original version is the parent node's version. Child trees allow for concurrent changes to be made to a node. In effect, the node is split and any concurrent changes are kept separate under their respective tree names.

creatree <module name> <parent tree> <parent r.d> <new tree>

<module name> name of an existing TCS module.

<parent tree> the parent node's tree name. If the parent tree is the main tree, 0 is submitted. The parent tree must already exist.

<parent r.d> the parent node's version number in the form of release.delta. The parent version must already exist.

<new tree> new tree's name
- a tree name may be from 1 to 31 characters in length.
- the first character must be alphabetic.
- characters other than the first can be any alphanumeric or special character.
- the tree name must not already exist. That is, tree names must be unique.

CREATREE EXAMPLES:

1. creatree program.c project_0 1.3 project_0a
   creates an empty tree, project_0a, whose parent node is "project_0 1.3".

2. creatree program.c 0 1.2 project_5
   creates an empty tree, project_5, whose parent node is version 1.2 in the main tree, 0.

GET

GET retrieves a version of a file stored within a TCS module. GET looks for a file called "<module name>.t," on the CURRENT WORKING directory, extracts the requested version, and places the version in a file called "t.<module name>" on the CURRENT WORKING directory. If a file by the name "t.<module name>" already exists, it is overwritten.

GET <module name> [-t <tree name> -v <r.d> -c -p -h]

<module name> the name of the TCS module on the current working directory.

OPTIONS:

Note: options are separated by white space and must NOT be catenated. The '-t' and '-v' options are separated from their arguments by white space.

-t <tree name> If submitted, a version from tree, "<tree name>," is retrieved. If omitted or if <tree name> is 0. then a version from the main tree is retrieved. If the tree is empty, then the parent version is retrieved.

-v <r.d> Requests a sepcific version within a tree. <r.d> is the version number where 'r' is the release level and 'd' is the delta number within the release. The '.' separates the release and delta number. The '.d' portion may be omitted. That is, a version number of the form '<r>' is also legal. In this case, the highest delta within release '<r>' is retrieved. If the '-v' option is omitted, GET retrieves the highest version in the tree.

-c Requests that the version retrieved be open

(reserved) for change purposes. GET logs the user's login ID, along with the time and date. If the version is already reserved for change, the user is so informed, shown the user ID, time and date of the reservation, and the GET fails. The GET also fails if the requested version is NOT the highest version in the tree.

-p Requests that the tree's path name be displayed. displays parent node names in the form:

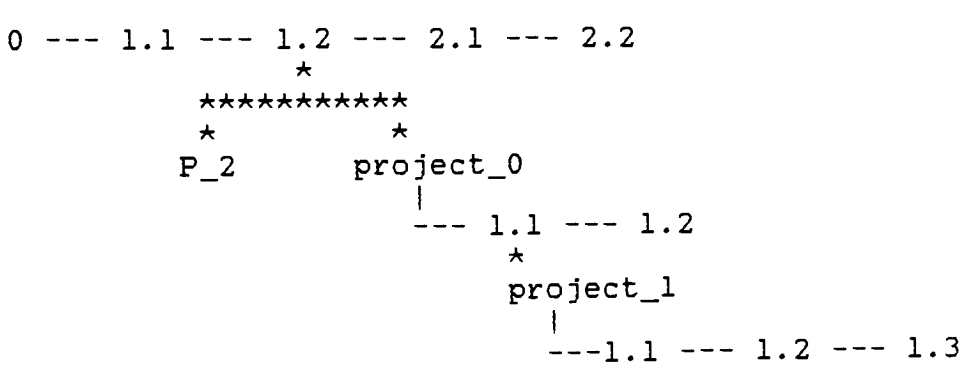
<tree name> <r.d>/

-h Requests the version's description information. GET displays the user ID of the user who created the version, creation time and date, and the description or comment for the version.

GET EXAMPLES:

A TCS module "program.c" exists in the file "program.c.t" on the current working diretory. GET will extract a version of "program.c" into a file called "t.program.c" on the current working directory.

Assume that the module's tree structure looks like:



The tree name precedes each delta chain.

1. get program.c
   or
   get program.c -t 0

   Verison 2.2 in the main tree (0) is retrieved.

2. get program.c -v 1
   Gets version 1.2 in the main tree.

3. get program.c -v 2.1
   Gets version 2.1 in the main tree.

4. get program.c -t project_1
   Gets version 1.3 in the tree, project_1

5. get program.c -t project_1 -v 1.1 -p
   Gets version 1.1 in tree, project_1 and also displays the pathname as:
   0 1.2/ project_0 1.1/ project_1 1.1

6. get program.c -c -t P_2
   Since tree, P_2, is empty, its parent version, 0 1.2 is retrieved. The -c causes a change reservation for tree, P_2.

PUT

Put a new version into the TCS module. Put creates a delta for the new verison and places the delta at the end of a tree's delta chain. It looks for a file given by the <module name> argument and a file called "<module name>.t" on the current working directory. The differences between the old and new version are obtained and the deltas are stored as the latest version in the tree. PUT cancels the change reservation. If the tree was not previously reserved for the change by the user executing the PUT or if there are no differences between the old and new version, the operation fails and no delta is created.

PUT prompts the user for a description (comment) of the new version. The description can be from 0 to 1000 characters in length. The prompt tells how many characters were entered so far with each line. A '.' on the first character, followed by a carriage return ends description prompting.

PUT stores the user's ID, time and date, and version description in the TCS modules' history file.

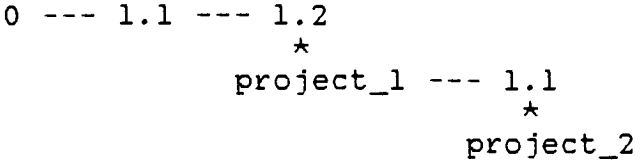
The new version is automatically numbered with the next delta number within the last release in the tree. When the '-r' option is submitted, PUT creates a new release for the tree, and the version's delta number is 1.

put <module name> <tree name> [-r]

<module name>	file name of the file containing the new version and the name of an existing TCS module file.
<tree name>	tree to which the new delta will be inserted. If <tree name> is 0, the delta is inserted at the end of the main tree's delta chain.
-r	creates a new release within the delta chain.

PUT EXAMPLES:

The user would like to enter a new version of program.c into the TCS module file program.c.t. Both files are on the current working directory. Assume that the TCS module's tree structure looks like:



1. put program.c 0
   The new version is 0 1.3. Tree 0 is the main tree.

2. put program.c project_2
   Since tree, project_2, is empty, the new version is created as version 1.1 of project_2.

3. put program.c project_1 -r
   The new version number is 2.1 for tree, project_1.

SHOW

Show or display information regarding the tree structure and/or about one or more trees in a TCS module. The information is written to the standard output.

show <what> <module name> [show function specific options]

<what> the desired SHOW function. Valid functions are: "tree" and "child".

<module name> TCS module name (on the current directory).

SHOW FUNCTIONS

SHOW TREE displays information regarding a tree or all trees.

show tree <module name> [-t <tree name> -apcdre]

-t <tree name>	tree name for which information is to be displayed. If this option is submitted, the -a option cannot also appear on the command line.
-a	displays the requested information for all trees.
-p	displays the tree's path name. If omitted, the tree and its parent node's name are displayed.
-c	tells whether or not the tree(s) is reserved for change. If the tree(s) is reserved, the reserver's user ID and the reservation time and date are displayed.
-r	displays release information. Shows the number of releases and the highest delta in each release.
-d	shows the last revision date for the tree(s).
-e	turns on the -p, -c, -r, and -d options. If present, the information provided by all

these options is displayed.

SHOW CHILD displays the names of a node's children trees.

show child ⟨module name⟩ [-t ⟨tree name⟩ -v ⟨r.d⟩ -c]

-t ⟨tree name⟩	the node's tree name whose children are to be listed. If omitted ⟨tree name⟩ defaults to the main tree.
-v ⟨r.d⟩	the node's version number within the tree. If omitted, ⟨r.d⟩ defaults to the highest version number.
-c	tells whether or not each child tree is reserved for change. The reservation's user ID, time and date are also displayed.

3.2 SUPER USER FUNCTIONS

XCANCEL Cancels the change reservation for a tree without checking the user ID. The form of this command is the same as that of the user command CANCELC.

PRINTMOD Prints the TCS module file containing the tree table release and change tables, and the deltas for all versions. Appendix B shows the format of PRINTMOD's output.

PRINTHIS Prints the TCS history file for a module. Prints the comment table entry and the description (comment) for each version in the module. Appendix C shows the format of PRINTHIS's output.

PRINTEMP Prints the temporary file created by PUT. This is the file containing line numbers for the "old version" retrieved during a PUT. The tree, release, and change tables are also displayed. Appendix D shows the format of PRINTEMP's output.

In order for PRINTEMP to function properly, the programmer must use this command in conjunction with a special testing version of the PUT.COM shell script. This test version saves the temporary line numbered file in a file called putsave/<module name>.l.

These functions are intended primarily for program debugging purposes. The print type commands display the contents of TCS files in readable form. Each produces a file on the current working directory. This file is input to MORE which displays the contents on the standard output. Output files are:

<module name>.p - PRINTMOD

<module name>.s - PRINTHIS

<module name>.r - PRINTEMP

CHAPTER 4

TCS INTERNAL DOCUMENTATION

The remainder of this paper deals with TCS's technical internals. The structure of the TCS module and history files, and the storage (PUT) and accessing (GET) algorithms are presented here.

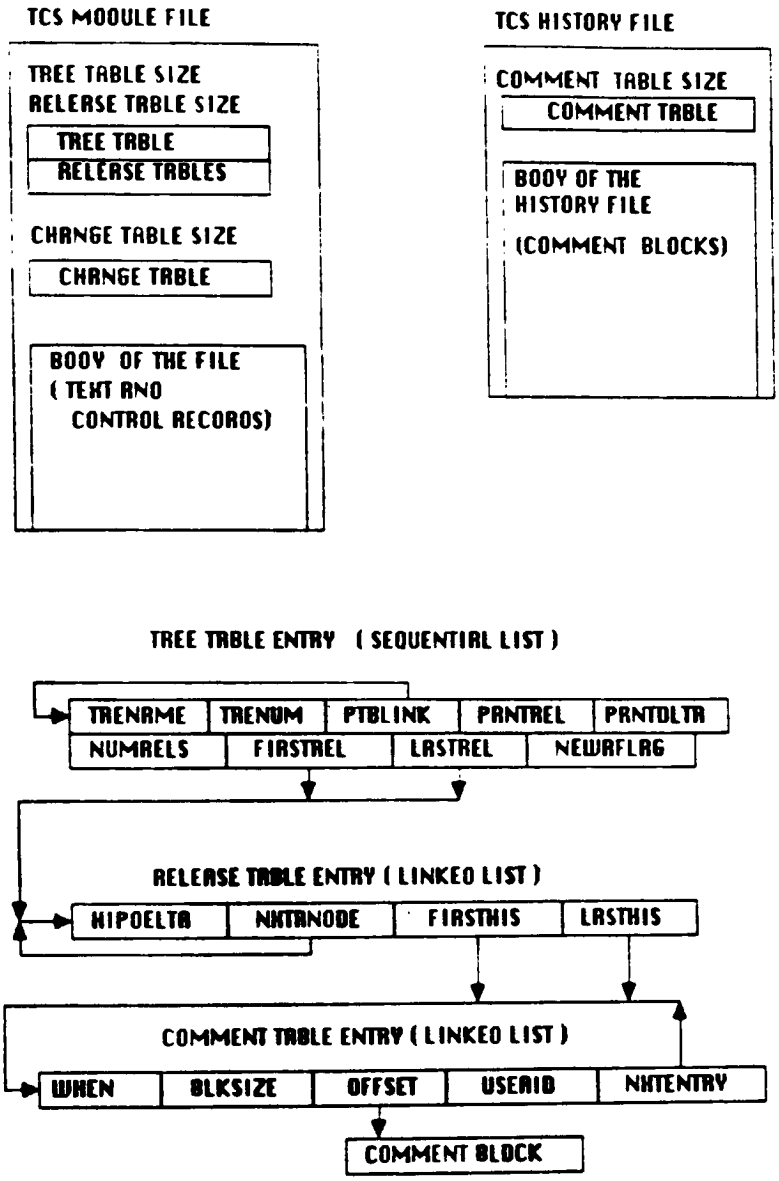
The reader will encounter extensive references to TCS programs and functions (subroutines). The TCS program is implemented in the C language. Each main program is in a ".c" file having the same name as the command that invokes it. Functions can be found in ".c" files having the same name as the functions.

Throughout this chapter, names of main programs, program variables, and TCS commands are shown in capital letters. Program functions names are of the form "<function>()." Arguments are not shown with any function name.

The author also makes references to the Appendices at the end of the paper. Appendix A contains data flow diagrams for all files used for TCS user commands. Appendices B, C, and D contain the formats of TCS files. Appendix E contains the source code for the TCS system. Appendix F contains the shell scripts for all TCS commands. Appendix G contains information for compiling and linking the TCS programs; and shell scripts used in compiling and linking.

The figures, 2a-2d show sample TCS files and the commands that created them. The last section, "A Sample TCS Session," is an extention of these figures. It shows a series of operations performed on the TCS module, "sample.1," and the all files used, modified or created by each TCS command issued. The reader may find it helpful to scan over the last section before reading the sections, "Accessing a Version," "Storing A Version," and "Accessing and Storing History Information."

Figure 2-0 contains an outline of the TCS module file and history file. This figure also shows the relationships among the tree, release, and comment tables discussed in the following sections. Arrows denote pointers to a table entry.



TCS FILES AND TABLE RELATIONSHIPS

FIGURE 2-D

4.1 THE TCS MODULE FILE

All information relating to the structure of the revision tree, the original version, deltas, and control records used to build versions are located in the TCS module file. The header information placed at the beginning of the file consists of the tree, release, and change tables and their sizes. Following the header information, is the body of the file where versions are stored.

The tree and release tables hold the information needed to construct the revision tree. These tables also contain information used to construct an internal data structure, known as the internal tree structure, that the accessing functions use during a retrieval operation.

4.1.1 The Tree Table

Each tree is assigned an internal tree number which the accessing algorithms use to identify the tree. The main tree's internal number is always one (1) and is assigned during the CREATMOD operation when the TCS module file is created. So, there is always one entry in the tree table. Stored with the tree table is the size of the table, corresponding to the number of trees

existing in the module. When a tree is added, this number is assigned to the new tree and then incremented by 1.

In the tree table, trees are kept sequentially by order of creation. The tree table is searched sequentially using the tree name as a key. The sequential nature of the table does not present a performance problem during any access operation since we do not expect that this table will be very large for any TCS module. The user will never notice the search time.

The structure for each tree table entry is found in the include file "tablestr.h" (struct treentry). The fields within each tree table entry are:

1. trename := The user assigned tree name

2. trenum := The internal tree number assigned at tree creation time.

3. ptblink := The Parent Node Back Link which is the index of the parent tree's entry in the tree table.

4. prntrel := The Parent's release number.

5. prntdlta := The Parent's delta number.

6. numrels := The number of releases in the tree.

7. firstrel := The index into the release table structure for the tree's first release.

8. lastrel := The index into the release table structure for the tree's last release.

9. newrflag := A new release flag that "PUT" uses in determining whether or not the delta being added should have a new release number.

4.1.2 The Release Tables

The release table is a collection of linked lists, one for each tree. This data structure contains information about each release in the tree. A node in a release table list consists of four fields containing the following information respectively (See "struct relnode" in "tablestr.h."):

1. hipdelta := The highest delta number in the release. This is initially 1 when a new release is installed and is updated on a PUT

that adds a delta to the release.

2. nxtrnode := A pointer into the release table array giving the index of the next release node for the tree.

3. firsthis := A pointer into the comment table (kept in the history file). This pointer is the index of the comment table entry for the first delta in the release.

4. lasthis := A pointer into the comment table for the last delta in the release.

The release table is kept linked so that releases can be dynamically added for any tree at any time.

4.1.3 The Change Table

The change table is a record of what trees are currently reserved for change. Each entry has three fields (See "struct change" in "tablestr.h"):

1. time := A value (obtained from the operating system) representing the date and time of the reservation.

2. trenum := The internal tree number of the tree whose last version is reserved for change.

3. userid := The login ID (obtained from the operating system) of the user who requested the reservation.

The change table is kept sequentially with reservations added at the end of the table. When PUT adds a delta to a reserved tree's last version, the tree's entry is deleted from the change table. This table expands and contracts as trees are reserved and deltas added. At any time, there is at most one entry for any tree. That is, only one person can reserve a tree for change at a time. The maximum size that this table can reach will be that of the tree table. So, like the tree table, since we do not expect that this table will grow very large, sequential searches go unnoticed by the user.

4.1.4 The Body Of The Module File

The text belonging to all versions is kept in the body of the file. Deltas are stored as a series of insertions and deletions to/from the versions that created them. Control records

surrounding one or more lines of text are used to identify which lines belong to each version. At the beginning of each line in the body of the file is a control character which tells the accessing routines whether the line that follows contains ASCII text or a control record.

Text is identified by a "t" control character, while control records are identified by an "I," "D," or "E" control character for insertion, deletion, or end control record respectively. An insert control record instructs the system that the text lines following it should be included for the version indicated on the record. The deletion does the opposite. It instructs the system to ignore text lines. Corresponding to each insertion and deletion control record is an "end" control record signalling that the effect of an "I" or "D" control should end. Each control record contains the tree number, release and delta number of the version to which it applies.

4.1.5 Illustrations

Figure 2-a shows a TCS module file "sample.1.t" containing one version (the original) of the user file "sample.1." In this figure, the

reader will note a "$ cat" UNIX command which printed the original version of the file "sample.1" The TCS "creatmod" command transformed the contents of this file into a TCS module and entered the user's first comment (shown after the "Enter comment..." prompt) into the history file (See figure 2-c). What follows is the contents of the TCS module file.

Figure 2-a.1 shows the module file after the user issued a "get" with the -c option. There is now one entry in the change table.

In figure 2-a.2, we see the first version editted under the "ex" editor. Then the "put" operation which creates the first delta.

Figure 2-b shows the second version of "sample.1" and the corresponding TCS module containing versions 1.1 and 1.2.

The TCS file contents were obtained in readable form from the "printmod" program. See Appendix B for the TCS module file and printmod output formats.

```
$ cat sample.1
This is a sample ASCII file. created using the ex editor.
It might have been a source code file.  But since TCS
doesn't know the difference and shouldn't care about
what's in the file, we'll keep this simple.
$ creatmod sample.1
Enter comment...
0 to 1000 characters

0: This is the original version of a sample file.
47: Its version name is 0 1.1 (the first version for tree 0 ).
10e: .

Comment entered....10e characters
sample.1.......TCS module has been created
$
$ printmod sample.1
tree table size = 2
0       1       -1      -1      -1
        1       0       0       0

        0       0       0       0
        0       0       0       0


release tables size = 1
1       0       0       0

change table size = 0

i 1 1.1
t This is a sample ASCII file. created using the ex editor.
t It might have been a source code file.  But since TCS
t doesn't know the difference and shouldn't care about
t what's in the file, we'll keep this simple.
e 1 1.1
$
```

Figure 2-a

```
$ printmod sample.1
tree table size = 2
0       1       -1      -1      -1
        1       0       0       0

        0       0       0       0
        0       0       0       0


release tables size = 1
1       0       0       0

change table size = 1
Sun Mar  2 11:55:18 1986
1       rcs0074

i 1 1.1
t This is a sample ASCII file. created using the ex editor.
t It might have been a source code file.  But since TCS
t doesn't know the difference and shouldn't care about
t what's in the file, we'll keep this simple.
e 1 1.1
$
```

Figure 2-a.1

```
$ cp t.sample.1 samole.1
$ ex sample.1
"samole.1" 4 lines, 205 characters
:0a
This line was added at the beginning of version 0 1.1.
.
:3
It might have been a source code file.  But since TCS
:s/.. ./.\

But since TCS
:$a

Ahb$ank line and this line were added at the end of 0 1.1.
.
:1,$
This line was added at the beginning of version 0 1.1.
This is a samole ASCII file, created using the ex editor.
It might have been a source code file.
But since TCS
doesn't know the difference and shouldn't care about
what's in the file, we'll keep this simple.

A blank line and this line were added at the end of 0 1.1.
:wq
"sample.1" 8 lines, 323 characters
$
$
$ put sample.1 0
Enter comment...
0 to 1000 characters

0:  This is the second version.
28: Atlwae add added at the beginning and end of version 0 1.1 and
91: a line in the middle was changed.
127: .

Comment entered....127 characters
'put' created: 0 1.2
$
```

Figure 2-a.2

```
$ get sample.1 -c
sample.1 0 1.1
4 lines retreived
$
$
```

```
$ prtdmac sample.1
tree table size   2
0       1       -1      -1      -1
        1       0       0       0

        0       0       0       0
        0       0       0       0


release tables size   :
2       0       0       1

change table size - 0

i 1 1.2
t This line was added at the beginning of version 0 1.1.
e 1 1.2
i 1 1.1
t This is a sample ASCII file. created using the ex editor.
d 1 1.2
t It might have been a source code file.  But since TCS
e 1 1.2
i 1 1.2
t It might have been a source code file.
t But since TCS
e 1 1.2
t doesn't know the difference and shouldn't care about
t what's in the file, we'll keep this simple.
i 1 1.2
t
t A blank line and this line were added at the end of 0 1.1.
e 1 1.2
e 1 1.1
```

Figure 2-b

4.2 THE TCS HISTORY FILE

All comments and records of who added a delta and when are kept in a separate file called "h.<module>." Like the module file, the history file consists of header information and the body where comments are stored.

4.2.1 The History File Header

At the beginning of the file is header information consisting of a structure containing the size of the comment table and the number of bytes in the body of the file. See "struct hsizes" in "histstr.h" for this structure. The header also contains the comment table which holds information about each comment in the body of the file. Each entry in the comment table consists of 5 fields (See "struct cominfo" in "histstr.h):

1. when := the date and time that the version (comment) was entered.

2. blksize := the number of bytes that the comment occupies in the file.

3. offset := the offset, in bytes, of the comment block from the beginning of the body of the history file.

4. userid := the login ID of the user who created the verison.

5. nxtentry := a pointer to the comment table entry for the next delta in the release.

Each time a delta is added to the module file, a new entry for the delta is inserted into the comment table. Since deltas are dynamically inserted, entries are kept linked. Each release has its own linked list of comment table entries. The release table entry holds the pointer (FIRSTHIS) to the comment table entry, corresponding to delta 1 of that release. Delta 2's entry is at the index given by the nxtentry field. Succeeding entries are found by following nxtentry links until a 0 in nxtentry is hit.

Since comments can be from 0 to 1000 characters in length, a "blksize" field is necessary to identify the comment size to the accessing routines. The "offset" field tells where the comment is found within the body of the file. Using these two values, the accessing routines can retrieve a comment in a random access fashion without having to read through the entire file to find a specific comment block.

4.2.2 The Body Of The History File

The user supplied comments follow one another in the body of the history files. These start immediately after the comment table. Comments are

added at the end of the file. The offset of a new comment is the HISTSZ value given in the size structure in the header. "HISTSZ" is updated by adding the number of bytes in the new comment. Therefore, "HISTSZ" always contains the offset from the beginning of the history body of the next available byte position.

Figures 2-c and 2-d show history files corresponding to the versions given in Figure 2-a and 2-b respectively. The "printhis" program displays the contents of the history file in readable format. Refer to Appendix C for the TCS history file and "printhis" output formats.

4.3 ACCESSING A VERSION

The main programs GET and PUT invoke the function getversn() to access a version. GET retrieves the version and delivers it to the user. PUT calls getversn() to retrieve the version to which a new delta is added (i.e., the "old version"). PUT, in addition, instructs getversn() to create a temporary module file containing line numbers for each line in the old version. A subsequent section explains how PUT uses these versions.

```
$ printhis sample.1
comment table size = 1
history size = 107

when..Sun Mar  2 11:48:47 1986
who..rcs0074     block size = 107          offset = 0       next entry index = 0
This is the original version of a sample file.
Its version name is 0 1.1 (the first version for tree 0 ).
-
```

Figure 2-c

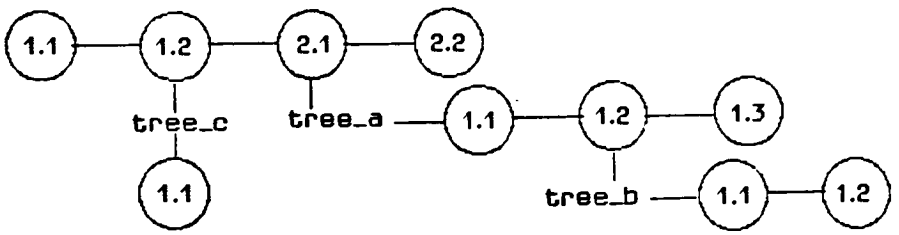
```
$
$ printhis sample.1
comment table size = 2
history size = 225

when..Sun Mar  2 11:48:47 1986
who..rcs0074     block size = 107          offset = 0       next entry index = 1
This is the original version of a sample file.
Its version name is 0 1.1 (the first version for tree 0 ).


when..Sun Mar  2 12:00:47 1986
who..rcs0074     block size = 128          offset = 107     next entry index = 0
This is the second version.
A line was added at the beginning and end of version 0 1.1 and
a line in the middle was changed.

$
```

Figure 2-d

4.3.1 Data Structures For The ACCESS Routines

The function getversn() decides which lines in the module file belong to the requested version by inspecting two internal data structures: "INTRESTR", the internal tree structure and "TRESTRTB", a cross reference into the internal tree structure. These structures provide a path of deltas to be applied in order to construct the

version. Function bldintst() builds the structures by starting with the tree, release, and delta number of the requested versions and following parent pointers in the tree table until the null parent (the main tree's parent) is reached. The tree number, the highest release to be included for the version, and an index into "INTRESTR" are recorded in TRESTRTB for each tree in the delta path. The index points to a list of highest deltas for each release that is to be included for a version. Figure 2-e shows a module's delta tree and the data in INTRESTR and TRESTRTB for the retrieval of version TREE_B 1.1.



INTRESTR

0	1	2	3
1	2	2	1

TRESTRTB

	TREENUM	TRESTNDX	NUMRELS
0	4	0	1
1	2	1	1
2	1	2	2

Figure 2-e

To reconstruct the version, getversn() must know that all deltas for the main tree (tree number = 1) up to and including 2.1, TREE_A (tree number = 2) up to and including 1.2 and TREE_B (tree number = 4) up to and including 1.1 are to be applied. When getversn() encounters and I or D control record, it calls setflag() to search the cross reference table for a match on the tree number. If the tree is found, then setflag() checks the number of releases, if the release number on the control record is less than or equal to the NUMRELS value in the table entry, then the delta is checked. Using the TRESTNDX index field, locates the tree's delta list in INTRESTR. Counting up to the release number in the control record, gets the exact location of the highest delta. getversn() compares delta numbers. If the delta number on the control record is less than or equal to the delta number in INTRESTR, the delta is applied. If any of these tests fail, the delta is not applied.

Refer to Figure 2-e for the following example. Given the control record: D 1 2.1, getversn() finds tree 1 at TRESTRTB[2]. The highest delta to be included for release 2 is at INTRESTR[3], whose value is less than or equal to the delta number 1 on the control record. Therefore, the delta text

corresponding to this control record is applied.

Since the text for any given delta doesn't always immediately follow its control records, getversn() uses another structure, the control queue in determing whether to retain a line of text as part of the requested version or skip it. See the include file queue.h for queue definitions and the file queue.c for the queue manipulation routines.

4.3.2 The Accessing Routines

getversn() reads the body of the module file line by line. If an insert (I) or delete (D) control record is encountered, a flag is set to "yes", "no" or "null" (YESFLAG, NOFLAG, and NULLFLAG values in defines.h). setflag() determines the flag setting by first deciding whether or not the delta should be applied. Flag is set to "yes" if the delta should be applied and the record is an insert (I). The flag is set to "no" if the delta should be applied and the control is delete (D) or if the delta should not be applied and the control is insert (I). The flag is set to "null" if the delta should NOT be applied and the control is delete (D). The tree, release, and

delta numbers, and the flag setting are entered at the front of the control queue. It is important to note that these entries are kept in chronological order with the latest at the head of the queue.

After an entry is made in the queue, getversn() (in routine i_d_rec()) sets an indicator "TAKELINE" based on the flag value just entered. TAKELINE tells getversn() whether to take or skip text lines. TAKELINE is set to the value in FLAG if flag is "yes" or "no." If flag is "null," then the queue is searched sequentially, beginning at the head, for the first non-null flag entry. TAKELINE is then set to this flag value.

When an end (E) control record is encountered, the queue is searched sequentially (beginning at the head) for the corresponding entry (tree, release, and delta numbers). The entry is removed from the queue and TAKELINE is reset based on the flag field of the entry at the head of the queue. The chronological nature of the queue along with the correspondence that PUT sets up between I, D and E control records guarantee that all searches on the queue are successful and that TAKELINE is appropriately set.

It would follow from all this, that getversn() merely needs to inspect TAKELINE when it encounters a TEXT (t) line. If TAKELINE is set to "yes", the line is written to the output file (the retrieved version). Otherwise, the line is ignored.

4.4 STORING A VERSION

The storage programs CREATMOD, PUT, and PUTDELTA, would have to surround text lines with control records so that accessing follows the description above. CREATMOD stores the original version (1 1.1) and initializes the tree and release tables. This operation is fairly simple : all text lines are preceded by an I 1 1.1 control record and followed by an E 1 1.1 control. Subsequent versions are stored in a multistep process (for the system, not the user) via the PUT.COM command file.

The main program, in PUT.C, builds the internal tree structure and calls getversn() to retrieve the "old" version, (to which the new delta will be added). This version is placed in a temporary file, invisible to the user. Put passes an indicator telling getversn() to add to another temporary file containing numbered text lines.

Those lines belonging to the old version are numbered in sequence starting with line one. Lines numbered with a 0 belong to other versions. PUTDELTA will later use this file to position new lines and control records in the updated module file. The line numbered file also contains the updated tree, release, and change tables. An Example of the temporary line numbered file is shown in the last section. This file was printed using the "printemp" command. See Appendix D for the PRINTEMP output format.

Before calling getversn(), Put has figured out the new version's tree, release, and delta numbers, removed the tree's entry in the change table, and entered the user's comment into the history file.

For the next step, PUT.COM passes the old and new versions to the UNIX Diff program. Diff determines the differences and places its output into a temporary file. Diff's output and the line numbered file are passed to the PUTDELTA program, which positions the new delta in the body of the new module file.

The Diff output is a series of additions, deletions, and change records. These records describe what lines in the old file need to be changed or deleted, and at what point lines would have to be inserted in order to recreate the new version from the old version. Figure 2-f shows a sample diff output.

```
$
$ cat /tmp/tcs.rcs/diff/sample.1
0a1
> This line was added at the beginning of version 0 1.1.
2c3,4
< It might have been a source code file.  But since TCS
---
> It might have been a source code file.
> But since TCS
4a7,8
>
> A blank line and this line were added at the end of 0 1.1.
$
```

Figure 2-f

Additions are preceded by an "a" type record containing the line number to which new lines are appended. An "a" record is followed by a series of ">" records containing the new lines. A "d" type record denotes that one or more lines are to be deleted. These are followed by "<" records containing the lines to be deleted. Changes to one or more lines are denoted by "c" records followed by "<" records showing the lines to be changed and ">" records showing what these lines are change to.

The format of the diff output lends itself well to processing by the PUTDELTA program. PUTDELTA interprets "a" and "d" records into TCS "i" and "d" records. Changed lines are stored as a deletion ("d") of the old lines and an insertion ("i") of the new lines. Using the storage scheme described in the following paragraphs, PUTDELTA ensures that the accessing function getversn() will find text lines in the order that is expects.

PUTDELTA reads the diff output line by line. When it encounters a diff "a", it reads the TCS temporary line numbered version up to and including the line after which the insert takes place. An "i" control record is written and the new lines are copied from the diff file as "t" type records, an "e" (end) control record is written, and the next diff line is read.

The function append() handles insertions. Given the line number from the diff "a" record, append works properly for insertions at the beginning, middle, or end of the version without making any special provisions for any of these modes of insertion. Insertions at the middle or end (i.e., after line 1) will show up immediately following the TCS line after which the insertion

takes place. If a "0a" diff line (append before the first line) is encountered, append() immediately writes out the insertion before writing any TCS lines, (see function rwfile() with a line number of 0 passed to it). "0a" operations do not necessarily place the inserted lines close to the first TCS line of the version but this doesn't matter; what is important is that the insertion is placed before the first line of the previous version. In fact, because of the sequential nature of the diff output, "0a" lines will always be processed first and the insertions that they cause always show up at the beginning of the body of the TCS file.

Deletions are handled in a similar manner but care must be taken when intervening "i" control records are encountered to ensure that inserts don't have an adverse effect during retrieval. For a deletion, PUTDELTA reads and writes lines from the line numbered file up to but NOT including the first line to be deleted. The "d" control record is then written out followed by TCS lines up to the next "i" control record or the last line to be deleted whichever comes first. An end control record is written out. If an "i" control caused the "e" control to be written, lines are read and

written from the numbered file until the next line to be deleted is located. A deletion control record is written and the program again writes TCS lines up to the next "i" control or the end of the deletion is encountered. Diff lines preceded by a "<" denoting the deleted lines are ignored.

A diff "c" type record causes a deletion followed by an insertion as described above. The function change(), calls delete() and passes delete() the line numbers on the "c" record. Then change() calls append() with a line number argument of 0 to insure that the insertion immediately follows the end control record for the deletion.

"i" records and their corresponding "e" records completely encompass their text lines. When storing new insertions, PUT does not need to worry about intervening "i" or "d" operations, that exist in the file. The correspondence with their existing "e" records remains intact. Furthermore, these records will be placed at the head of the "getversn()" control queue after the new insertion. Inserts and deletes that belong to the version will be applied. Those that do not, will be ignored.

New deletions, however, don't cause lines from earlier versions, along the delta path, to be applied. If a deletion would encompass inserts from earlier versions, setflag() would think that these inserts should be applied and return a "yes" flag, regardless of whether or not the insert should be applied. Any intervening insert from a previous version would then be placed in front of the deletion entry in the control queue, causing the insert to be included. Rather than complicating setflag() to handle this situation, it is simpler for PUTDELTA to ensure that there are no intervening "i" records for a delete.

Refer to the sample session in the last session for an example of how The module file looks after the addition of version 1.3 in the main tree (tree, 0). This version deleted lines from the two previous versions. This part of the TCS session begins with the command line:

```
$ get sample.l -v 1.2 -c -p
```

Note that, by this time, there is also a branch of version 0 1.1 which was created and updated before version 0 1.3 was added.

4.5 STORING AND ACCESSING HISTORY INFORMATION

The section entitled "The TCS History File" gave an overview of the history file and described how information is ordered within the file. This section deals with the details of storing and accessing a version's comment, the login ID of the user who created the version, and the time of creation.

The main programs CREATMOD and PUT store history information. CREATMOD calls creathis() to create the history file for a new module, initialize the comment table with an entry for version 0 1.1, and store the first comment. PUT calls updhist() to update the history file whenever a delta is added to the module.

The function creathis() and the PUT program invoke the UNIX C function getlogin() to retrieve the user's login ID. They invoke the function getcommt() to get the user submitted comment. The function getcommt() prompts the user for lines of comment with the number of characters read so far and then invokes the C function getline() to retrieve a line from the standard input. As lines are read, getcommt() keeps a count of characters in the comment. The user signals the end of the comment by typing a line whose only character is

".". After this, getcommt() marks the end of the comment with the null character "\0" and returns the character count which includes the "\0".

PUT then passes the user ID, comment size (character count), and the comment to updhist() which updates the comment table and stores the new comment at the end of the file. updhist() invokes updcomtb() to add an entry at the end of the comment table and link the new entry to that of the previous delta within the release. updcomtb() then invokes the UNIX C function time() and fills in the comment table entry's "when" field with a long integer representing the date and time the entry is made. The "offset" field, showing the position of the new comment within the body of the history, is filled in with the value of the HISTSZ field in the size structure stored at the beginning of the file. HISTSZ always points to the next available offset value; so HISTSZ is updated by adding the number of characters in the new comment (given in the variable BLKSIZE).

Once the new comment table entry has been made and the new comment table and history sizes have been returned, updhist() writes the new header information (sizes and table) to the updated

history file, reads and writes out all comments existing in the old history file, and finally writes the new comment.

CREATMOD also calls updcomtb() which is general enough to handle an empty table. CREATMOD then merely writes out the size structure, the comment table containing a single entry, and the initial comment to the newly created history file.

The GET and SHOW programs read the information stored in the history file and write it in readable form to the standard output. When the user issues a GET command with the -h option, the GET program calls gethis() to retrieve the user ID, time, and comment for the requested version.

GET passes the delta number, the release table, and a pointer to the release table entry for the version. gethis() passes these onto gcomindx() which returns the index of the comment table entry for the version. The function gcomindx() gets the index into the comment table for the first delta in the release from the FIRSTHIS field of the version's release table entry. It traverses the comment table by following the indices in the comment table's NXTENTRY field until it reaches the entry for the delta in question.

Having gotten the index into the comment table, gethis() reads the "offset" value for the comment and passes it to the C function fseek() to position the history file pointer at the beginning of the comment. A simple fread() call, using the BLKSIZE field for the number of characters to read in, retrieves the comment into an internal I/O buffer. Since the comment has a "\0" at the end, it is now easily printed as a single character string by invoking printf(). The user ID is output directly from the USERID field, since this is already in ASCII character format. The "when" value is translated from long integer form into readable form by the UNIX C function ctime() and then output.

The SHOW TREE command with the -d option will display the last revision date for a tree. The SHOW program calls prtinfo() to output the date and time from the comment table entry of the highest version in the tree's main branch. Prtinfo() gets the release table index for the version from the LASTREL (last release) field of the tree table entry. The comment table index is found in the LASTHIS (last history, i.e., of the highest delta in the release) field of the version's release table node. The C function ctime() is invoked to

translate the comment table's "when" field and the date and time are output.

4.6 THE USER INTERFACE

The discussion of the TCS internals has so far dealt with the lowest level storage and access mechanisms. Temporary files, tables, linked lists, etc. come into play during the execution of a TCS command. This is all invisible to the user and rightly so. Furthermore, while the user is aware of the existence of a TCS file on the current working directory, he/she need not be concerned that this is actually an archive file containing two separate files, the module and history file.

Issuing a TCS command causes the execution of a shell script. Each command has its own corresponding shell script (see Appendix F). Each TCS command name is an alias for the the command's script. After checking that a TCS file exists on the current working directory, a TCS script will create a temporary directory: /tmp/tcs.$$. UNIX guarantees that this directory is unique for the process. The script copies the TCS archive file to /tmp/tcs.$$ and changes to this directory where all work is performed. Some commands, GET and PUT for

instance, require subdirectories of /tmp/tcs.$$ to keep temporary files separate. The script takes care of these cases also. At the completion of the command, all temporary directories are removed.

Once on the temporary directory, the script will extract the individual TCS files from the archive file and invoke the command's corresponding program. All arguments are passed to the program. Each program will interpret the argument string, perform user error checks, convert user input into a form that the low level routines understand, and then invoke the appropriate routine based on the input argument.

The commands GET, SHOW, and CREATREE have the most complex argument strings, requiring a reference to a specific version, either by default or explicitly stated on the command line. For instance, the command GET TEST1 will, by default, retrieve the highest version in the main tree.

Access routines like bldintst() and getversn() do not need to know about the argument string. What they need is the internal integer tree, release and delta numbers. Furthermore, low level routines like these assume that their input data is correct and perform no error checks on user input.

Whatever checks the low level routines perform are strictly to catch programming errors.

Input error checks and access to internal version numbers are performed by a group of functions that are linked using the script USERINTR.OLINK. This script produces an object file that is later linked to the GET, SHOW, and CREATREE programs. The programs for other commands explicitly link selected functions from this user interface group (see the link scripts in Appendix G).

4.7 A SAMPLE TCS SESSION

On the following pages is a terminal session demonstrating several TCS operations on the "sample.1" file after versions 0 1.1 and 0 1.2 were added. The contents of all files are printed after each operation using the PRINTMOD, PRINTHIS, and PRINTEMP programs. The diff output is also shown for a "put" operation.

The last operations are "show tree" and "show child" after version 0 1.3 was added.

```
$ createe samcle.1 0 1.1 tree_e
'createe'....Tree, tree_a, created.  It's parent is: 0 1.1
$
$ get sample.1 -t tree_a -c
sample.1, tree_a
Tree is empty.
Parent retrieved: 0 1.1
4 lines retrieved
$
$ cp saeaple.1 sample.1
ex sample.1
"sample.1" 4 lines, 209 characters
:2
It might have been a source code file.  But since TCS
:1
This is a sample ASCII file, created using the ex editor.
:d
It might have been a source code file.  But since TCS
:wq
"sample.1" 3 lines, 151 characters
$
$
$ put sample.1 tree_a
Enter comment...
0 to 1000 characters

0: This is the first tree.
24: To create this version, I deleted the first line from
78: the parent version 0 1.1.
104: wq
107: .

Comment entered....107 characters
'put' created: tree_a 1.1
$
$
$ patnttmp/tcs.rcs/diff/sample.1
1d0
< This is a sample ASCII file, created using the ex editor.
$
```

```
$
$ printemp sample.1
tree_a 1.1

2 1.1
tree table size = 2
0       1       -1      -1      -1
        1       0       0       0

tree_a  2       0       1       1
        1       1       1       0

        0       0       0       0
        0       0       0       0


release tables size = 2
2       0       0       1
1       0       2       2

change table size = 0

i 1 1.2
t 0  This line was added at the beginning of version 0 1.1.
e 1 1.2
i 1 1.1
t 1  This is a sample ASCII file, created using the ex editor.
d 1 1.2
t 2  It might have been a source code file.  But since TCS
e 1 1.2
i 1 1.2
t 0  It might have been a source code file.
t 0  But since TCS
e 1 1.2
t 3  doesn't know the difference and shouldn't care about
t 4  what's in the file, we'll keep this simple.
i 1 1.2
t 0
t 0  A blank line and this line were added at the end of 0 1.1.
e 1 1.2
e 1 1.1
$
```

```
$
$ printmod sample.1
tree table size = 2
0       1       -1      -1      -1
        1       0       0       0

tree_a  2       0       1       1
        1       1       1       0

        0       0       0       0
        0       0       0       0


release tables size = 2
2       0       0       1
1       0       2       2

change table size = 0

i 1 1.2
t This line was added at the beginning of version 0 1.1.
e 1 1.2
i 1 1.1
d 2 1.1
t This is a sample ASCII file. created using the ex editor.
e 2 1.1
d 1 1.2
t It might have been a source code file.  But since TCS
e 1 1.2
i 1 1.2
t It might have been a source code file.
t But since TCS
e 1 1.2
t doesn't know the difference and shouldn't care about
t what's in the file, we'll keep this simple.
i 1 1.2
t
t A blank line and this line were added at the end of 0 1.1.
e 1 1.2
e 1 1.1
$
```

```
$
$ printhis sample.1
comment table size = 0
history size = 340

when..Sun Mar  2 11:48:40 1986
who..rcs0074     block size   107        offset = 0       next entry index = 1
This is the original version of a sample file.
Its version name is 0 1.1 (the first version for tree 0 ).


when..Sun Mar  2 12:00:47 1986
who..rcs0074     block size   128        offset = 107     next entry index = 0
This is the second version.
A line was added at the beginning and end of version 0 1.1 and
a line in the middle was changed.


when..Sun Mar  2 12:10:35 1986
who..rcs0074     block size = 108        offset = 235     next entry index   0
This is the first tree.
To create this version, I deleted the first line from
the parent version 0 1.1.
wq

$
$ get sample.1 -v1122

sample.1 0 1.2
8 lines retreived
$ $
$
$ cp tsample.1 sample.1
$ ex sample.1
"sample.1" 8 lines, 323 characters
:1,2d
It might have been a source code file.
:wq
"sample.1" 6 lines, 210 characters
$
$ cat sample.1
It might have been a source code file.
But since TCS
doesn't know the difference and shouldn't care about
what's in the file, we'll keep this simple.

A blank line and this line were added at the end of 0 1.1.
$
```

```
$ get sample.1 -v 1.2 -c -p
sample.1
0 1.2
8 lines retrieved
$
$ ep t.sample.1 sample.1
$ ex sample.1
"sample.1" 8 lines, 323 characters
:1,2d
It might have been a source code file.
:wq
"sample.1" 6 lines, 210 characters
$ cat sample.1
It might have been a source code file.
But since TCS
doesn't know the difference and shouldn't care about
what's in the file, we'll keep this simple.

A blank line and this line were added at the end of 0 1.1.
$
$
$ put sample.1 0
Enter comment...
0 to 1000 characters

0: This version deleted lines from previous versions.
51: It is an illustration of how TCS handles deletions
102: with intervening insert records.
135: .

Comment entered....135 characters
'put' created: 0 1.3
$
$
$ patch tmp/tcs.rcs/diff/sample.1
1,2d0
< This line was added at the beginning of version 0 1.1.
< This is a sample ASCII file, created using the ex editor.
$
```

```
$
$ printemp sample.1
0 1.2

1 1.2
tree table size = 2
0       1       -1      -1      -1
        1       0       0       0

tree_a  2       0       1       1
        1       1       1       0

        0       0       0       0
        0       0       0       0


release tables size = 2
3       0       0       2
1       0       2       2

change table size = 0

i 1 1.2
t 1  This line was added at the beginning of version 0 1.1.
e 1 1.2
i 1 1.1
d 2 1.1
t 2  This is a sample ASCII file. created using the ex editor.
e 2 1.1
d 1 1.2
t 0  It might have been a source code file.  But since TCS
e 1 1.2
i 1 1.2
t 3  It might have been a source code file.
t 4  But since TCS
e 1 1.2
t 5  doesn't know the difference and shouldn't care about
t 6  what's in the file, we'll keep this simple.
i 1 1.2
t 7
t 8  A blank line and this line were added at the end of 0 1.1.
e 1 1.2
e 1 1.1
$
```

```
$
$ printmod sample.1
tree table size   3
0         1        -1       -1       -1
          1        0        0        0

tree_a    2        0        1        1
          1        1        1        0

          0        0        0        0
          0        0        0        0


release tables size = 2
2         0        0        2
1         0        2        2

change table size = 0

i 1 1.2
d 1 1.3
t This line was added at the beginning of version 0 1.1.
e 1 1.2
e 1 1.3
i 1 1.1
d 2 1.1
d 1 1.3
t This is a sample ASCII file, created using the ex editor.
e 1 1.3
e 2 1.1
d 1 1.2
t It might have been a source code file.  But since TCS
e 1 1.2
i 1 1.2
t It might have been a source code file.
t But since TCS
e 1 1.2
t doesn't know the difference and shouldn't care about
t what's in the file, we'll keep this simple.
i 1 1.2
t
t A blank line and this line were added at the end of 0 1.1.
e 1 1.2
e 1 1.1
$
```

```
$
$ orinthis sample.1
comment table size = 4
history size = 479

when..Sun Mar  2 11:48:47 1986
who..rcs0074    block size = 107        offset = 0      next entry index = 1
This is the original version of a sample file.
Its version name is 0 1.1 (the first version for tree 0 ).


when..Sun Mar  2 12:00:47 1986
who..rcs0074    block size = 128        offset = 107    next entry index = 3
This is the second version.
A line was added at the beginning and end of version 0 1.1 and
a line in the middle was changed.


when..Sun Mar  2 12:10:35 1986
who..rcs0074    block size = 108        offset = 235    next entry index = 0
This is the first tree.
To create this version, I deleted the first line from
the parent version 0 1.1.
wq


when..Sun Mar  2 12:23:14 1986
who..rcs0074    block size = 136        offset = 343    next entry index = 0
This version deleted lines from previous versions.
It is an illustration of how TCS handles deletions
with intervening insert records.
```

```
$ show tree sample.1 -ae

0 1.1/ tree_a 0.0
Number of releases: 1
1.1
NOT reserved for change
Last revision date:  Sun Mar  2 12:10:35 1986

0 0.0
Number of releases: 1
1.2
NOT reserved for change
Last revision date:  Sun Mar  2 12:23:14 1986
$
$
$ show child sample.1 -t0 -v1.1
0 1.1 child trees:
tree_a
1 child trees
$
$
```

CHAPTER 5

BIBLIOGRAPHY

[1] Walter F. Tichy, "RCS - A System for Version Control," Software - Practice and Experience, Vol. 15(7), pp. 637-654, July 1985

[2] Softool Corp., Change and Configuration Control (CCC) User's Manual, Jan 1985

[3] Softool Corp., "SoftAware," p. 1, Vol. 1, No. 1, Jan 1985

[4] Digital Equipment Corp., User's Introduction to VAX DEC/CMS, Field Test Draft, Oct 1984

[5] David B. Leblang and Robert P. Chase, Jr., "Computer-Aided Software Engineering in a Distributed Workstation Environment," Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Pratical Software Development Environments, Apr 1984

[6] Edward H. Bersoff, "Elements of Software Configuration Management," IEEE Transactions on Software Engineering, Vol. se-10, No. 1, Jan 1984

[7] Appollo Computer, Inc., Domain Software Engineering Environment (DSEE) Reference Manual, 1984

[8] Bell Telephone Laboratories, UNIX System User's Manual System V, Jan 1983

[9] B. R. Rowland and R. J. Welsh, "The 3B20D Processor and DMERT Operating System: Software Development System," The Bell System Technical Journal, Vol. 62, No. 1, Jan 1983

[10] Software Maintenance And Development Systems, Inc., Aide-de-Camp Software Engineering Environment User's Guide, 1983

BIBLIOGRAPHY

[11] Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," Proceedings of the 6th Int'l Conf. on Software Engineering, pp 58-67, Sep 1982

[12] Alan L. Glasser, "The Evolution of a Source Code Control System," Proceedings of the Software Quality and Assurance Workshop, Vol. 3(5), pp. 122-125, Nov 1978.

[13] Evan 1. Ivie, "The Programmers Workbench - a Machine for Software Development," Communications of the ACM, Vol 20, No. 10, Oct 1977

[14] Marc J. Rochkind, "The Source Code Control System," IEEE Transactions on Software Engineering, Vol. se-1, No. 4, pp. 364-370, Dec 1975

APPENDIX A

TCS FILES

The diagrams in this appendix represent a data flow of input and output files for TCS user commands, in alphabetical order. Program names are given within circles. File names shown before an arrow pointing to a circle are input files. Arrows pointing to a file name suggest an output file.

All files NOT on the current working directory are shown with a path name preceding the file name. The following key contains a definition of terms used in the diagrams.

UNIX program names:

ar r := archive program invoked with
the r, replace, option

ar x := archive program invoked with
the x, extract, option

cp := copy program

diff := differences program

Terms in filenames:

<mod> := the native file name, user chosen, for a file under TCS control

t.<mod> := the TCS module file for <mod>
or
the file delivered to the user on GET.

h.<mod> := the TCS history file for <mod>

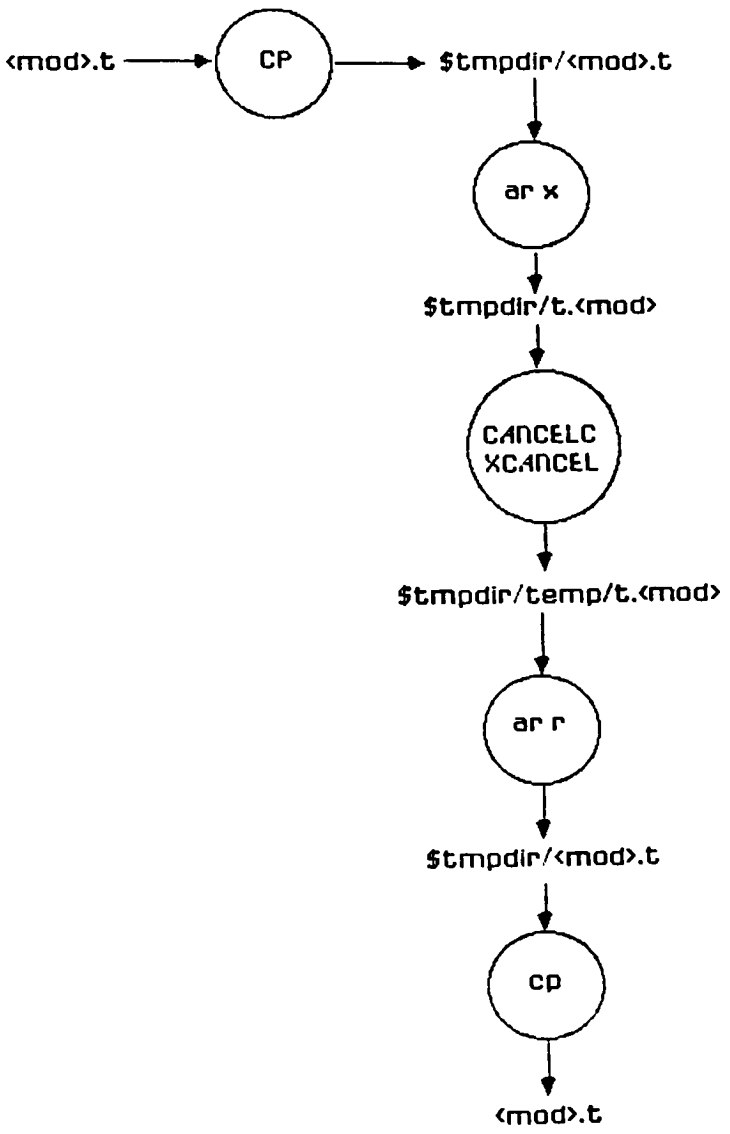
<mod>.t := the TCS archive file containing the module file (t.<mod>) and the history file (h.<mod>)

$tmpdir := A symbol representing the tempoary directory, /tmp/tcs.$$

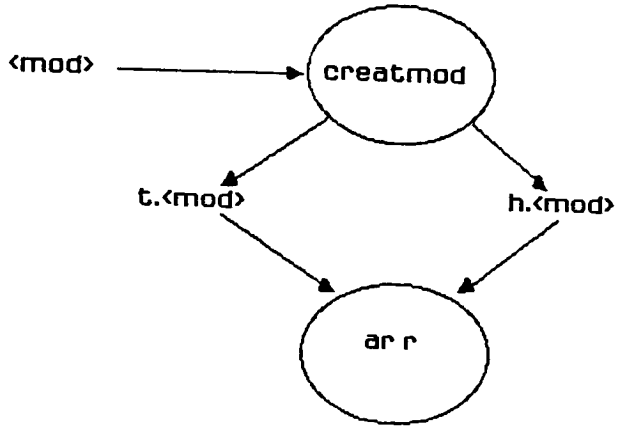
tmpfile2 := an intermediate (temporary) file

histmp := an intermediate history file

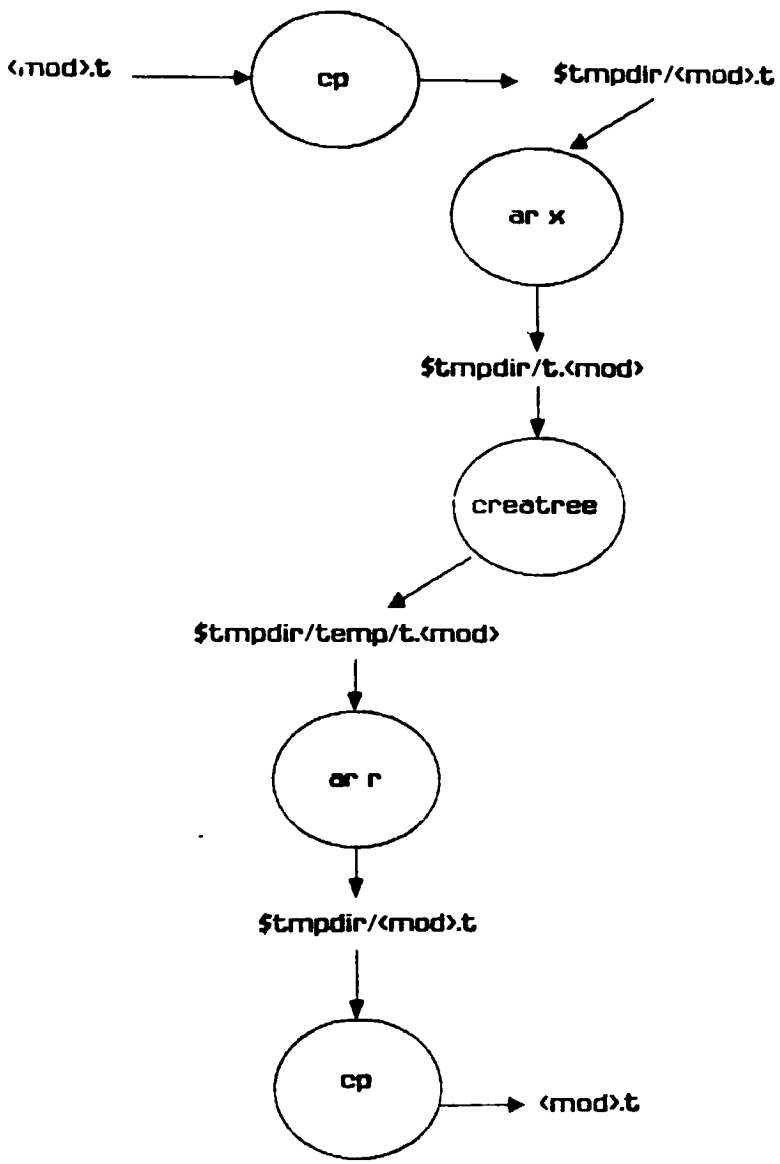
CANCELC and XCANCEL



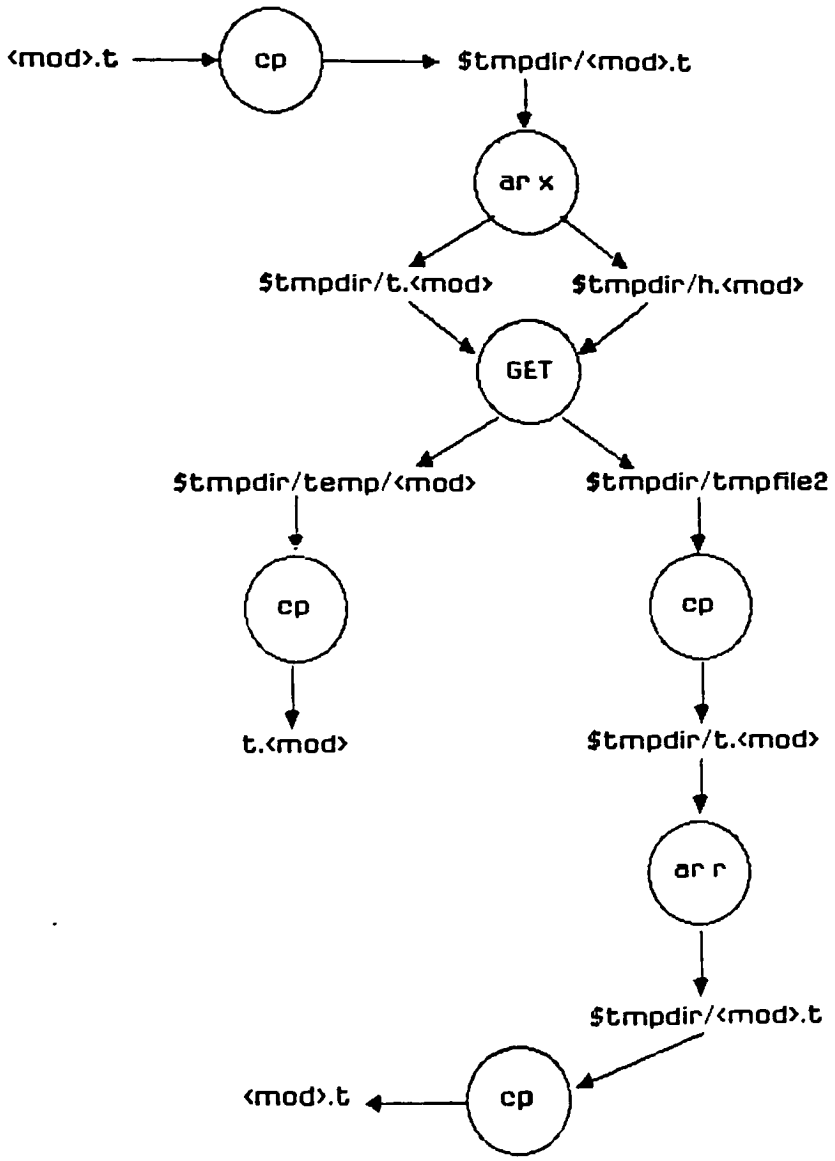
CREATMOD



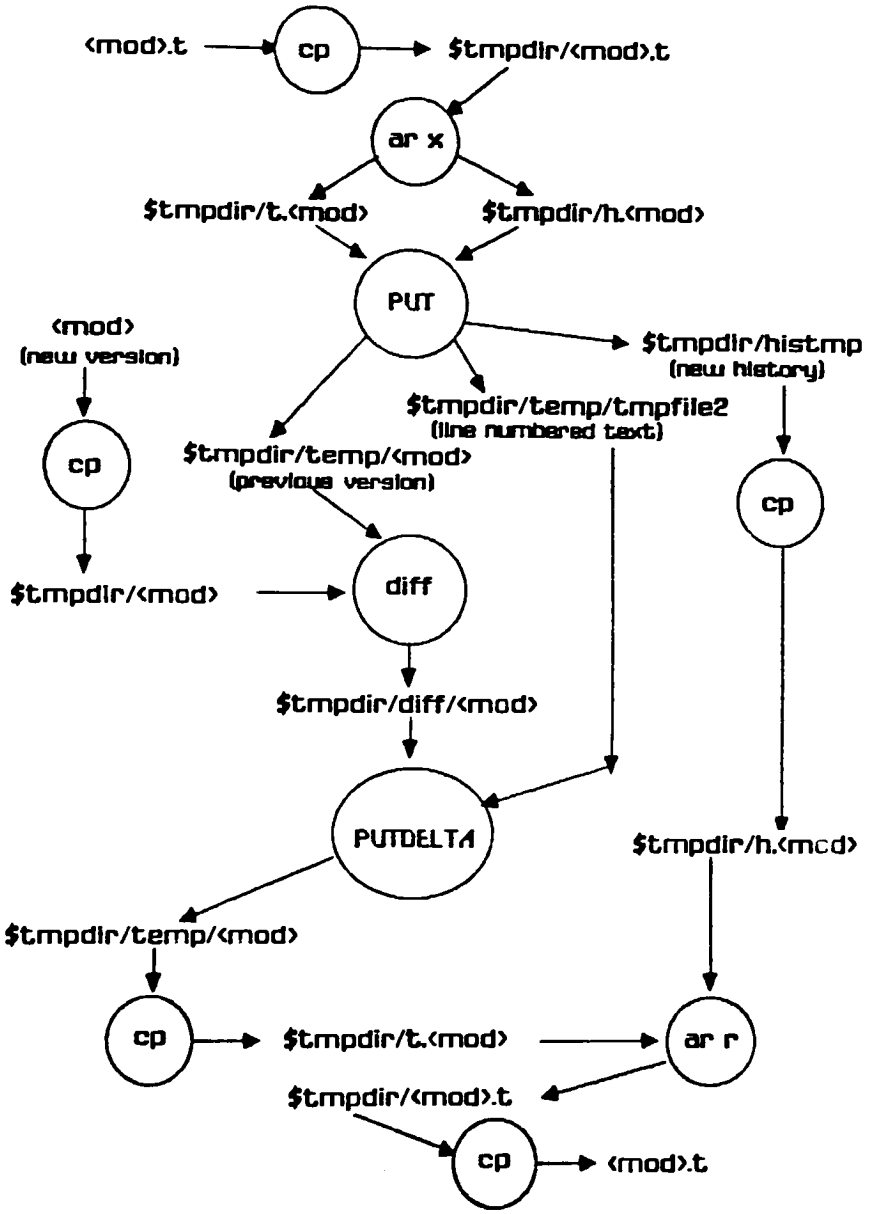
CREATREE

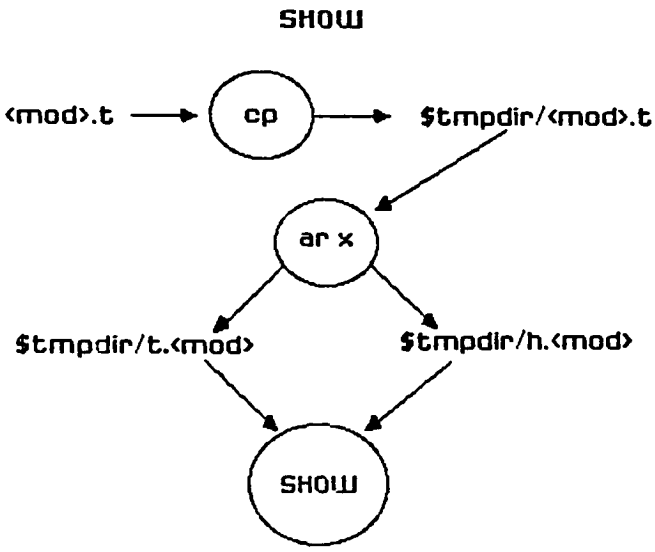

<mod>.t
cp
$tmpdir/<mod>.t
ar x
$tmpdir/t.<mod>
createe
$tmpdir/temp/t.<mod>
ar r
$tmpdir/<mod>.t
cp
<mod>.t

GET


<mod>.t
cp
$tmpdir/<mod>.t
ar x
$tmpdir/t.<mod>
$tmpdir/h.<mod>
GET
$tmpdir/temp/<mod>
$tmpdir/tmpfile2
cp
cp
t.<mod>
$tmpdir/t.<mod>
ar r
$tmpdir/<mod>.t
<mod>.t
cp

PUT


<mod>.t
cp
$tmpdir/<mod>.t
ar x
$tmpdir/t.<mod>
$tmpdir/h.<mod>
PUT
<mod>
(new version)
$tmpdir/histmp
(new history)
$tmpdir/temp/tmpfile2
(line numbered text)
cp
$tmpdir/temp/<mod>
(previous version)
cp
$tmpdir/<mod>
diff
$tmpdir/diff/<mod>
PUTDELTA
$tmpdir/h.<mod>
$tmpdir/temp/<mod>
cp
$tmpdir/t.<mod>
ar r
$tmpdir/<mod>.t
cp
<mod>.t


SHOW
<mod>.t
cp
$tmpdir/<mod>.t
ar x
$tmpdir/t.<mod>
$tmpdir/h.<mod>
SHOW

APPENDIX B

TCS MODULE FILE FORMATS

Section 1 contains the format of a TCS module file. Section 2 contains the format of the PRINTMOD program output. Refer to include files tablestr.h and tabledef.h for the fields within a table entry and the table definitions. The include file contlstr.h shows the contents of a control record.

The function readtbls() reads the tree, release and, change tables into memory. The function writble() writes them to a file. See the main program in printmod.c for more details.

B.1 THE TCS MODULE FILE

```
-----------------------------------------------
TSIZE    integer >= 2    number of entries in
                         the tree table
RSIZE    integer >= 1    number of nodes in
                         the release table
-----------------------------------------------
/* The tree table */
/* For each tree table entry up to TSIZE
   entries */
```

```
TRENAME  character[]     tree name
                         '\0' for the last
                         tree table entry.
TRENUM   integer >= 1    internal tree number
PTBLINK integeer >= -1   pointer to parent tree.
                         -1 for the main tree.
PRNTREL  integer >= -1   parent node's release
                         -1 for the main tree.
PRNTDLTA integer >= -1   parent node's delta
                         -1 for the main tree.
NUMRELS  integer >= 0    number of releases in
                         the tree. 0 for an
                         empty tree
FIRSTREL integer >= -1   pointer to the tree's
                         first release node.
                         -1 for an empty tree.
LASTREL  integer >= -1   pointer to the tree's
                         lastrel release node.
                         -1 for an empty tree.
NEWRFLAG integer 0 or 1  new release flag
-------------------------------------------------
/* Release Table */
/* For each release table entry up to
   RSIZE entries */
HIPDELTA  integer >=1    highest delta in the
                         release
NXTRNODE  integer >=0    pointer to the next
                         node in a tree's
                         release list.  0 marks
                         the end of the release
                         list for the tree.
FIRSTHIS integer >=-1    pointer to the release's
                         first comment table
                         entry. -1 for an empty
                         tree.
LASTHIS integeer >= -1   pointer to the release's
                         last comment table
                         entry. -1 for an empty
                         tree.
-------------------------------------------------
CSIZE    integer >=0     number of entries in the
                         change table
-------------------------------------------------
/* Change Table */
/* For each change table entry up to CSIZE
   entries */
TIME     long integer    a value representing
                         the time and date,
                         obtained from the
                         operating system
TRENUM   integer >= 1    internal tree number
USERID  character[]      User ID, obtained from
```

```
                the operating system.
-------------------------------------------------
/* Body of the module file */
-------------------------------------------------
/* Control Record */
CONTLCHR  character  'i','d', or 'e'     control
                                        character
TREENUM   integer >= 1     internal tree number
RELNUM    integer >= 1     release number
DELTANUM  integer >= 1     delta number
-------------------------------------------------
/* Text Record */
CONTLCHR  character 't'    text control character
TEXT      character[]      text string containing
                  any ASCII character.
                  The new line character,
                  '\n' marks the end of the
                  string.
-------------------------------------------------
```

B.2 PRINTMOD OUTPUT

Refer to the table fields in the TCS module file format for the meaning of the fields for the table entries below.

```
-------------------------------------------------
/* Tree Table size */
Tree Table Size = TSIZE
-------------------------------------------------
/* For each tree table entry up to TSIZE entries */
TRENAME  TRENUM  PTBLINK PRNTREL PRNTDLTA
 NUMRELS FIRSTREL LASTREL NEWRFLAG
-------------------------------------------------
/* Release Table size */
release table size = RSIZE
-------------------------------------------------
/* For each release table entry up to RSIZE
   entries */
HIPDELTA  NXTRNODE FIRSTHIS LASTHIS
-------------------------------------------------
/* Change Table size */
change table size = CSIZE
-------------------------------------------------
/* For each change table entry up to CSIZE
   entries */
TIME     TRENUM  USERID
-------------------------------------------------
/* body of the file */
-------------------------------------------------
/* 'i', 'd' or 'e' Control Record */
CONTLCHR  TREENUM RELNUM.DELTANUM
-------------------------------------------------
/* 't' Text Record */
t TEXT
-------------------------------------------------
          .
          .
          .
-------------------------------------------------
/* 'e' End Control Record */
e TREENUM RELNUM.DELTANUM
-------------------------------------------------
```

APPENDIX C

THE TCS HISTORY FILE FORMAT

Section 1 contains the format of a TCS history file. Section 2 contains the format of the PRINTHIS program output. Refer to include files histstr.h and histdefs.h for the fields within a table entry and the table definitions. See the main program in printhis.c for more details.

C.1 THE TCS HISTORY FILE

```
-----------------------------------------------------
/* size record */
COMTBSZ  integer >= 1    comment table size
HISTSZ   integer >= 1    number of bytes in the
                         history body
-----------------------------------------------------
/* Comment Table */
/* For each comment table entry up to COMTBSZ
   entries */
WHEN      long integer   a value representing the time
                         and date that the entry was
                         made. Obtained from the
                         operating system.
BLKSIZE   integer >= 1   number of bytes in the
                         comment; includes the '\0'
                         character that marks the end
                         of the  comment string.
OFFSET    integer >= 0   number of bytes from the
                         beginning of the history
                         body; the comment block's
```

```
                offset.
USERID  character[]     User login ID; obtained from
                the operating system.
NXTENTRY integer >=0    pointer to the next entry
                within the comment table
                for a release. 0 marks the
                end of a comment list.
-------------------------------------------------------
/* History Body */
/* For each comment block up to COMTBSZ comments */
TEXT     character[BLKSIZE]
-------------------------------------------------------
```

C.2 PRINTHIS OUTPUT FORMAT

Refer to the Section 1 for the meaning of the fields and variables mentioned in this section.

```
-------------------------------------------------------
/* Comment table and history body sizes */
comment table size = COMTBSZ
history size = HISTSZ
-------------------------------------------------------
/* Comment table entries and comments */
/* For each comment block up COMTBSZ blocks */

when..WHEN
who..USERID block size = BLKSIZE  offset = OFFSET
next entry index = NXTENTRY
/* Comment */
TEXT
-------------------------------------------------------
```

APPENDIX D

PRINTEMP OUTPUT FORMAT

The PRINTEMP program displays the contents of the temporary file created by the PUT program. This file is similiar to the TCS module file. But it contains a few extra records in the beginning and text records have integer line numbers preceding the text strings. See the main program in printemp.c for more details.

Refer to Appendix B for the meaning of record fields and variable values mentioned here. The variable LINNO shown in a Text type record is an integer value (>= 0). If a line of text belongs to the version (old version) to which PUTDELTA will add the new delta, then LINNO contains the text's line number within the old version. Otherwise, LINNO is 0.

PRINTEMP OUTPUT FORMAT

```
------------------------------------------------------
/* New Version Identification Record */
TREENAME  RELNUM.DELTANUM
/* New Version's Control Record Information */
TRENUM  RELNUM.DELTANUM
------------------------------------------------------
/* Updated Tree Table size */
Tree Table Size = TSIZE
------------------------------------------------------
/* Updated Tree Table */
/* For each table entry up to TSIZE entries */
TRENAME  TRENUM  PTBLINK PRNTREL PRNTDLTA
 NUMRELS FIRSTREL LASTREL NEWRFLAG
------------------------------------------------------
/* Release Table size */
release table size = RSIZE
------------------------------------------------------
/* Updated Release Table */
/* For each release table entry up to RSIZE
   entries */
HIPDELTA  NXTRNODE FIRSTHIS LASTHIS
------------------------------------------------------
/* Updated Change Table size */
change table size = CSIZE
------------------------------------------------------
/* Updated Change Table */
/* For each change table entry up to CSIZE
   entries */
TIME     TRENUM  USERID
------------------------------------------------------
/* body of the file */
------------------------------------------------------
/* 'i', 'd' or 'e' Control Record */
CONTLCHR  TREENUM RELNUM.DELTANUM
------------------------------------------------------
/* 't' Text Record */
t LINNO TEXT
------------------------------------------------------
          .
          .
          .
------------------------------------------------------
/* 'e' End Control Record */
e TREENUM RELNUM.DELTANUM
------------------------------------------------------
```

APPENDIX E

SOURCE CODE

The source code for all TCS programs and functions are in files whose names are of the form <command>.c for main programs and <function>.c for functions called within the program. "<command>" is the name of the TCS command that invokes the program. ".h" files are include files.

All TCS source code is on the $home/source directory. To change to this directory, use the alias: tcsrc (tcsrc = cd $home/source).

```
tcsrc
$
$ owd
/a3/s1/rcs0074/source
$
$ ls *.h
contlstr.h    filedefs.h    histstr.h     shotrdef.h    showextn1.h   tablestr.h
defines.h     histdefs.h    idinfo.h      showextn.h    tabledef.h

$ pwd
/a3/s1/rcs0074/source
$
$ ls *.c
append.c      delete.c      getversn.c    prtrinfo.c    show.c        versubm.c
bldintst.c    difflt.c      gopt.c        put.c         showchil.c    writabls.c
cancelc.c     fndendch.c    gpthname.c    putdelta.c    showtree.c    wrtchgtb.c
change.c      fndtrdnm.c    intrd.c       queue.c       streeopt.c    wrtcontl.c
chkopen.c     gcomindx.c    openall.c     readtble.c    tcsopt.c      wrtrtble.c
chkrsvc.c     get.c         parcopta.c    readtbls.c    updcomtb.c    xcancel.c
closeall.c    getcommt.c    printbl.c     rwbody.c      updhist.c
convatob.c    gethird.c     printemp.c    rweof.c       updtbls.c
creathis.c    gethis.c      printhis.c    rwfile.c      updtrtbl.c
creatmod.c    getreptr.c    printmod.c    rwhistf.c     util.c
creatree.c    getrindx.c    prtrel.c      setflag.c     verifyrd.c
```

```
/* control character definitions and control record structure definitions */


struct control{
        int treenum;    /* tree number */
        int relnum;     /* release number */
        int deltanum;   /* delta number */

};

#define inscontl 'i'    /* insert control character */
#define delcontl 'd'    /* delete control character */
#define endcontl 'e'            /* end control character */
#define txtcontl 't'    /* text control character */
```

```
#define arg_size 35
#define trnamesz 33
#define num_opt 5
#define true 1
#define false 0
#define treearg 0       /* index of the tree argument indicator in opt_indc */
#define versarg 1 /*index of version argument indicator in opt_indc */
#define changarg 2 /* index of change argument indicator in opt_indc */
#define pathnarg 3 /* index of path name argument in opt_indc */
#define histarg 4 /* index of history argument in opt_indc */
#define reldelim '.'  /* release delimitor in the version name argument */
#define nullchar '\0' /* null character */
#define maxtrees 100  /* maximum # of entries in the tree tables */
#define maxrnode 1000 /* maximum # of release nodes */
#define numvers 5000 /*maximum # of versions */
#define modsize 100  /* max. characters in TCS module name */
#define mainame "0"     /* main tree''s name */
#define maintnum 1      /* main tree's number */
#define mainprnt -1     /* main tree's parent back link, release, & delta*/
#define initnrel 1      /* initial release number for non-empty trees */
#define inithdel 1      /* initial highest delta for non-empty trees */
#define buffsize 1001   /* i/o buffer size */
#define difflnsz 132 /* 'diff' file line size */
#define yesflag 1       /* flag meaning that a delta should be applied*/
#define noflag  2       /* flag that a delta should not be applied */
#define nullflag 3      /* null flag */
#define timesz 26       /* number of characters in o.s. time info */
#define useridsz 32     /* number of characters in the user id */
```

```
/* filename array definitions */

char modname[modsize] = "t.";  /* TCS directory/module name */
char tempfile[modsize] = "temp/";
char tmpfile2[] = "temp/tmpfile2";
char histfile[modsize] = "h.";
char histmp[] = "histmp";       /* temporary history file */
char diffile[modsize] = "diff/"; /* diff output file */
```

```
/* tables, table sizes, and total blocksize in history file */

struct cominfo comtable[numvers];       /* comment table */

struct hsizes hstsizes; /* history size and comment table size */
```

```
/* Structure of the comment table in the module's history file */

struct hsizes{
        int comtbsz;    /* comment table size */
        int histsz;     /* size of the body of the history file */
};

struct cominfo{
        long when;      /* when the change was made */
        int blksize;    /* version's comment block size */
        int offset;     /* offset of the comment block from the beginning
                           of history body */
        char userid[useridsz];  /* who made the change */
        int nxtentry;   /* index of next comment table entry for the release*/
};
```

```
/* "defines.h should be included before this one is */
/* Contains the buffer definition for the version identification string */

                /* 'arg_size is defined in "defines.h"
                The given size allows room for the tree name,
               version name, a few blanks, and the end of string character */
#define idbufsiz  2 * arg_size + 5
char idbuff[idbufsiz];
```

```
        /* definitions for the 'show tree' function */

        /* maximum optional arguments */
#define sho_argn 8
        /* optional argument flags */
#define t_opt showind[0]        /* tree name option flag */
#define a_opt showind[1]        /* show ALL trees option flag */
#define p_opt showind[2]        /* pathname option flag */
#define r_opt showind[3]        /* show releases option flag */
#define c_opt showind[4]        /* change table entry option flag */
#define d_opt showind[5]        /* last modified date option flag */
#define e_opt showind[6]        /* turn on p,r,c,d options */
```

```
        /* External definitions used by 'show functions' */
        /* Tree table, release tables, change table, and
           comment table pointers */
extern struct treentry treetble[maxtrees];
extern struct relnode rtbnodes[maxrnode];
extern struct change chgtble[maxtrees];
extern struct cominfo comtable[numvers];
        /* comment table size info */
extern struct hsizes hstsizes;
        /* tree, release, and change table sizes */
extern int tretblsz, reltblsz, chgtbsz;
```

```
        /* External definitions used by 'show functions' */
        /* Tree table, release tables, change table, and
           comment table pointers */
extern struct treentry *treetble;
extern struct relnode *rtbnodes;
extern struct change *chgtble;
extern struct cominfo *comtable;
        /* comment table size info */
extern struct hsizes hstsizes;
        /* tree, release, and change table sizes */
extern int tretblsz, reltblsz, chgtbsz;
```

```
/* table definitions */

struct treentry treetble[maxtrees];     /* tree table */

struct relnode rtbnodes[maxrnode];      /* release tables */


struct change chgtble[maxtrees];        /* change table */

        /* internal tree structure */
int intrestr[maxrnode];
        /* cross reference table for the internal tree structure */
struct refs trestrtb[maxtrees];

int tretblsz;   /* tree table size */
int reltblsz;   /* release table size */
int chgtbsz;    /* change table size */
```

```
struct treentry { /* tree table entry */
        char trename[arg_size]; /* tree name */
        int  trenum;            /* tree number */
        int  ptblink;           /* parent tree index into tree table */
        int prntrel;    /* parent's release number */
        int prntdlta;   /* parent's delta number */
        int  numrels;           /* number of releases */
        int  firstrel;          /* index - tree's first release in rel. table */
        int lastrel;            /* index- tree's last release in rel. table */
        int newrflag;   /* new release flag */
};


struct relnode {        /* release table node */
        int hipdelta;   /* highest propagating delta */
        int nxtrnode;   /* index of next release node */
        int firsthis;   /* index of first history node for the release */
        int lasthis;    /* index of last history node for the release */
};


struct change{  /* change table entry */
        long time;      /* date and time change was cataloged */
        int trenum;     /* tree number */
        char userid[useridsz];  /* user ID; who made the change. */
};



struct refs {   /* entry in the cross reference table for the */
                /* internal tree structure */
        int  trenum;    /* tree number */
        int  trestndx;  /* index into internal tree structure for trenum */
        int  numrels;   /* number of releases to be included in the search */
};
```

```
/* Install an 'i' (insert) type delta into the new module file */

#include (stdio.h)
#include "defines.h"
#include "contlstr.h"

char
*append( after, contlptr, fp_diff, fp_temp, fp_out, diffbuff, iobuff )

        /*input arguments */
int after;      /* line after which lines are appended */
struct control *contlptr; /*pointer to the control record for the new version*/
FILE *fp_diff; /* diff file */
FILE *fp_temp; /* temporary module file with line numbered text lines */
        /* output arguments */
FILE *fp_out; /* temporary new module file */
char *diffbuff; /* contains a line from the diff file */
char *iobuff; /* contains lines from the temporary module file */
{
        int writincl; /* 'write including line' indicator */
        char *diffeof; /* end of diff file indicator */
        char type;

                /* Read and write lines from the temporary file, up to and
                   including 'after' line */
        writincl = 1;
        rwfile( after, writincl, fp_temp, fp_out, iobuff );

                /* Write the new 'insert' control character and record */
        type = inscontl;
        wrtcontl( type, contlptr, fp_out );

                /* Read the lines to be inserted from the 'diff' file and
                   write them out */
        diffeof = fgets(diffbuff, difflnsz, fp_diff );
        while ( diffbuff[0] == ')'  &&  diffeof != NULL )
        {
                        /* Write the line to the new module file */
                        /* Write out the text control character */
                putc( txtcontl, fp_out);
                fputs( diffbuff+2, fp_out );
                        /* Read the next 'diff' file line */
                diffeof = fgets( diffbuff, difflnsz, fp_diff );
        }

                /* Write out the end control character and record */
        type = endcontl;
        wrtcontl( type, contlptr, fp_out );

        return (diffeof);
}
```

```
/* Build the internal tree structure, and the cross reference table
   into the internal tree structure. These structures are used by
   'getversn'( get_version ) and 'put'.
*/

#include "defines.h"
#include "tablestr.h"

bldintst( treeptr, relnum, deltanum, reltable, treetble, intrestr, trestrtb)

        /* input arguments */
struct treentry *treeptr;       /* tree pointer */
int relnum;                     /* release number */
int deltanum;                   /* delta number */
struct relnode reltable[];      /* release tables */
struct treentry treetble[];     /* tree table */

        /* output arguments */
int intrestr[];         /* internal tree structure */
struct refs trestrtb[]; /* cross reference table */

{
        int tstrindx;   /* index into the internal tree structure */
        int relindx;    /* index into the release tables */

                /* traverse the tree table for parent nodes. When
                'treeptr' points below the tree table,the main tree's
                release info has been entered into the internal tree
                structure */

        tstrindx = 0;   /* initialize the index into the I.T.S. */

        while (treeptr )= treetble)
        {
                        /* place the tree number into the cross reference
                           table */
                trestrtb->trenum = treeptr->trenum;
                        /* place the number of releases to be included
                           for the tree into the cross reference table */
                trestrtb->numrels = relnum;
                        /* place the cross reference into the c.r. table*/
                trestrtb->trestndx = tstrindx;

                        /* get the index of the first release node for
                           the current tree */
                relindx = treeptr->firstrel;
                        /* enter release info for all releases that
                           precede 'relnum' into the internal tree
                           structure */
                while ( --relnum )
                {
                        /* place the highest delta for the current release
                           into the I.T.S. and point to the next available
                           position in the I.T.S. */
```

```
                        /* get the index of the next release node*/
                    relindx = reltable[relindx].nxtrnode;

                        /* keep track of the index into the I.T.S.*/
                    tstrindx++;
                }

                    /* place the delta for the last release to be
                       included for the tree into the I.T.S. */
                    /* and point to the next available position in
                       the I.T.S. */
                *intrestr++ = deltanum;
                    /* keep track of the index into the I.T.S. */
                tstrindx++;

                    /* get the parent's release nuumber */
                relnum = treeptr->prntrel;
                    /* get the parent's delta number */
                deltanum = treeptr->prntdlta;
                    /* get the pointer to the parent's tree
                       table entry */
                treeptr = treetble + (treeptr->ptblink);
                    /* point to the next available position in
                       the cross reference table */
                trestrtb++;
            }
                /* mark the end of the cross reference table*/
            trestrtb->trenum = -9999;
}
```

```
/* Cancel the change reservation for a tree:  remove the tree's entry from the
   change table */

/* USAGE:     cancelc (module name) (tree name) */

#include (stdio.h)
#include (time.h)
#include "defines.h"
#include "tablestr.h"
#include "contlstr.h"
#include "tabledef.h"
#include "filedefs.h"


#define error_1 "Module %s does not exist\n", argv[1]

#define error_2 "Cannot open %s\n", tempfile

#define msg_1  "Change reservation cancelled for: %s, %s\n", argv[1], argv[2]


main( argc, argv )

int argc;
char *argv[];
{
        struct change *chgptr, *chkopen();
        char *getlogin();
        struct treentry *treeptr, *getreptr(); /* tree pointers */
        char iobuff[buffsize];  /* i/o buffer */

        FILE *fp_mod, *fp_temp, *fopen();

                /* check that the required number of arguments have been
                   submitted */
        if ( argc ( 3 )
        {       printf("module or tree name argument missing\n");
                exit(1);
        }


                /* Open the module file */
        strcat( modname, argv[1]);
        if ( (fp_mod = fopen( modname, "r")) == NULL )
        {       printf( error_1 );
                exit (1);
        }

                /* Open the temporary new module file */
        strcat( tempfile, argv[1]);
        if ( (fp_temp = fopen( tempfile, "w" )) == NULL )
        {
                printf ( error_2 );
                exit (1);
```

```
        /* read the tables from the TCS file */
    readtble ( fp_mod, &tretblsz, &reltblsz, treetble, rtbnodes, &chgtbsz,
        chgtble);

        /* get the tree's pointer into the tree table */
        /* If the tree name is not found in the table, print error*/
    if ( !(treeptr = getreptr(argv[2],treetble)) )
    {   printf("%s...tree does not exist\n", argv[2]);
        exit (1);
    }

        /* Check that the tree is open for change. If it is also
           get it's pointer into the change table */
    if ( !( chgptr = chkopen( treeptr, chgtble, chgtbsz )) )
    {   printf("%s is not open for change\n", argv[2]);
        exit (1);
    }

        /* Check the user ID */
    if ( strcmp( chgptr->userid, getlogin() ) != 0 )
    {   printf("Access denied; %s has %s open for change\n",
                chgptr->userid, treeptr->trename );
        exit (1);
    }

        /* The new release indicator is no longer needed.
           Turn it off */
    treeptr->newrflag = 0;

        /* Write the tree and release tables */
    wrtrtble( &tretblsz, &reltblsz, treetble, rtbnodes, fp_temp );

        /* Write out the change table, deleting the tree's entry */
    wrtchgtb( fp_temp, chgtble, chgtbsz, chgptr );

        /* Write out the body of the module */
    rwbody( fp_mod, fp_temp, iobuff );

        /* Tell the user that the change reservation has
           been cancelled */
    printf( msg_1 );
    exit (0);
} /* end main */
```

```
/* Install a 'd' (delete) and an 'i' (insert) delta into the new module
   file. Change lines in the old version for the new version */

#include (stdio.h)
#include "defines.h"
#include "contlstr.h"

char
*change( from, to, contlptr, fp_diff, fp_temp, fp_out, diffbuff, iobuff )

        /*input arguments */
int from; /* first line to be deleted for the new version */
int to; /* last line to be deleted for the new version */
struct control *contlptr; /*pointer to the control record for the new version*/
FILE *fp_diff; /* diff file */
FILE *fp_temp; /* temporary module file with line numbered text lines */
        /* output arguments */
FILE *fp_out; /* temporary new module file */
char *diffbuff; /* contains a line from the diff file */
char *iobuff; /* contains lines from the temporary module file */
{
        char *diffeof; /* end of diff file indicator */
        char *delete(), *append();
        int after;

                /* Delete the lines to be changed */
        diffeof = delete( from, to, contlptr, fp_diff, fp_temp,
                        fp_out, diffbuff, iobuff);
                /* Insert the new lines */
        after = 0; /* insert before the next module line */
        diffeof = append( after, contlptr, fp_diff, fp_temp,
                        fp_out, diffbuff, iobuff );

        return (diffeof);
}
```

```
/* Search the change table for the tree number given by 'treeptr' */

#include "defines.h"
#include "tablestr.h"

struct change
*chkopen ( treeptr, chgtble, numentry )

struct trentry *treeptr;        /* pointer to tree table entry */
struct change chgtble[];        /* change table */
int numentry;   /* number of entries in the change table */

{
        /* If the tree number is found in the change table, return the
           pointer to the change table entry. Otherwise return, 0 */

        while ( numentry-- )
        {
                if ( chgtble->trenum == treeptr->trenum )
                        return (chgtble);
                chgtble++;
        }

        return (0);
}
```

```
/* check a character string for a reserved character */

char
*chkrsvc( string, reservedc)

char *string;   /* character string */
char reservedc[];       /* array of reserved characters */

/* The function returns the pointer to the reserved character in the string
   if one is found.  Otherwise, it returns a zero */
{
        int i;  /* traverses the reserved character array */
        char c; /* character under examination */

        while ( c = *string )
        {
                for ( i = 0; reservedc[i] != '\0'; i++ )
                        if ( c == reservedc[i] )
                                return (string);
                string++;   /* point to the next character in the string */
        }

        return (0);
}
```

```
/* close all files used by 'put' */

#include (stdio.h)

closeall( fp_mod, fp_old, fp_temp, fp_hist, fp_htmp )

FILE *fp_mod;   /* TCS module file */
FILE *fp_old;   /* file containing the old version to be used by 'diff'*/
FILE *fp_temp;  /* file containing the line numbered text to be used
                   by 'putdelta' */
FILE *fp_hist;  /* history file */
FILE *fp_htmp;  /* temporary new history file */

{
        fclose(fp_mod);
        fclose(fp_old);
        fclose(fp_temp);
        fclose(fp_hist);
        fclose(fp_htmp);
}
```

```
/* convert an ascii string to integer */
/* the function assumes that the string contains numeric ascii characters */
/*  with the exception of the delimiter                                    */


convatob( strgbegn, strgend )

char *strgbegn;  /* pointer to the first character in string */
char *strgend;     /* pointer to the string delimiter          */

{
        /*  'ptr' is used to traverse the string character by character */
        char *ptr;

         /* 'tempatob' is used to calculate the integer value          */
        int tempatob;

        for ( tempatob = 0, ptr = strgbegn; ptr != strgend; ptr++)
                tempatob = tempatob * 10 + (*ptr - '0');
        return (tempatob);
}
```

```
/* Create the history file for the new module */

#include (stdio.h)
#include (time.h)
#include "defines.h"
#include "tablestr.h"
#include "histstr.h"
#include "histdefs.h"

creathis( treeptr, reltable, fp_hist )

        /* input arguments */
struct treentry *treeptr;       /* tree's tree table entry pointer */
        /* output arguments */
struct relnode reltable[];      /* release tables */
FILE *fp_hist;  /* history file */
{

        int blksize;    /* comment size in characters */
        char *userid, *getlogin();      /* user id */
        char comment[buffsize]; /* Buffer into which comment is read */

                /* Get the comment from the user */
        blksize = getcommt( comment, buffsize );

                /* get the user id */
        userid = getlogin();

                /* initialize the comment table size and the body of
                   history file size */
        hstsizes.comtbsz = 0;
        hstsizes.histsz = 0;

                /* Update the comment table */
        if ( !updcomtb( userid, blksize,treeptr, reltable, &hstsizes, comtable))
                return (0);

                /* Write out the new comment table and history sizes */
        fwrite( &hstsizes, sizeof(struct hsizes), 1, fp_hist );
                /* Write out the new comment table */
        fwrite( comtable, sizeof(struct cominfo), hstsizes.comtbsz, fp_hist );

                /* Write out the new comment */
        fwrite( comment, sizeof(*comment), blksize, fp_hist );

        return (1);
}
```

```
/* Create a module under TCS */
/* The Command Line:
        creatmod (module)

        (module) is the name of the input text file.
        There must be at least one line in the file for the system to work
        correctly.
*/

#include (stdio.h)
#include "defines.h"    /* constants definitions */
#include "contlstr.h"   /* control record structure and definition */
#include "tablestr.h"
#include "tabledef.h"
#include "filedefs.h"
FILE *fopen(), *fp_in, *fp_out, *fp_hist;
char iobuff[buffsize];  /* i/o buffer */
char type;      /* control record type */
struct control contlrec = { 1, 1, 1 }; /* control record for the main tree */

main( argc, argv )

int argc;       /* number of arguments */
char *argv[];   /* argument pointers */

{

                /* make sure that a module name was submitted */
        if (argc < 2 )
        {
                printf( "module name missing\n" );
                exit (1);
        }

                /* open the input file, if the file exists */

        if ( ( fp_in = fopen( argv[1], "r" ) ) == NULL )
        {
                printf("cannot open %s \n", argv[1] );
                exit (1);
        }

                /* concatenate the module name with the directory
                name for TCS */
        strcat( modname, argv[1] );
                /* make sure that the TCS module doesn't already
                   exist */
        if ( (fp_out = fopen(modname, "r")) != NULL )
        {
                printf( "%s...Module already exists\n", modname);
                exit (1);
        }

                /* open the output file */
        if ( ( fp_out = fopen( modname, "w") ) == NULL )
```

```
                printf( "cannot open %s\n", modname );
                fclose(fp_in);
                exit (1);
        }

                /* Open the new history file */
        strcat( histfile, argv[1] );
        if ( (fp_hist = fopen( histfile, "w" )) == NULL )
        {
                printf( "'creatmod' cannot open %s\n", histfile );
                fclose(fp_in, fp_out);
                exit (1);
        }
                /* initialize the tree, release, and change tables */
        initabls();

                /* Create the new history file */
        if (!creathis( treetble, rtbnodes, fp_hist ) )
                exit (1);

                /* write the size of the tables and the tables out
                   to the TCS file */
        writabls( fp_out, &tretblsz, &reltblsz, treetble, rtbnodes,
                &chgtbsz, chgtble );

                /* write out the insert record control character */
        type = inscontl;        /* fill in the control character */
        fwrite( &type, sizeof(type), 1, fp_out );
                /* write the Insert Control Record for the main tree.
                  The record has been initialized, above, with the main
                  tree's info */
        fwrite( &contlrec, sizeof(struct control), 1, fp_out );

                /* Read the text records, line by line and write them
                   out. Each line output is preceeded by the text
                   control character */
        while ( writetxt() );

                /* write out the end control record */
        type = endcontl;        /* fill in the end control character */
                /* write the end control character */
        fwrite( &type, sizeof(type), 1, fp_out );
                /* write the control structure for the main tree */
        fwrite( &contlrec, sizeof(struct control), 1, fp_out );

                /* tell user that module has been created */
        printf( "%s......TCS module has been created\n", argv[1] );

        clfile(fp_in, fp_out, fp_hist);
        exit (0);
}

initabls()

        /* initialize the tree, release, and change tables and their sizes. */
```

```
{
        struct treentry *treeptr;

        treeptr = treetble;
        strcpy( treeptr->trename, mainame ); /* fill in main tree name*/
        treeptr->trenum = maintnum;  /* fill in the main tree number */
        treeptr->ptblink = mainprnt; /* fill in parent back link */
        treeptr->prntrel = mainprnt; /* fill in parent release number */
        treeptr->prntdlta = mainprnt; /* fill in parent delta number */
        treeptr->numrels = initnrel; /* fill in number of releases */
        treeptr->firstrel = 0;  /* first release node index */
        treeptr->lastrel = 0; /* last release node index */
        treeptr->newrflag = 0; /* new release indicator */

        treeptr++;      /* point to the next entry in the tree table */
        strcpy( treeptr->trename, ""); /* mark end of table */

                /* set up the release table entry */
        rtbnodes[0].hipdelta = initnrel; /* highest release */
        rtbnodes[0].nxtrnode = 0;  /* mark end of main's release table */
        rtbnodes[0].firsthis = -1; /* indicate an empty history table */
        rtbnodes[0].lasthis = -1;

        tretblsz = 2;   /* initial tree table size */
        reltblsz = 1;   /* initial release tables size */
        chgtbsz = 0;    /* initial change table size */

}



writetxt()
{
        char *fgets();

                /* read a line of text from the original file */
        if ( fgets( iobuff, buffsize, fp_in ) == NULL )
                return (0);

        type = txtcontl;        /* set up the text control character */
        fwrite( &type, sizeof(type), 1, fp_out);

                /* write the line to the TCS file */
        fputs( iobuff, fp_out);
        return (1);
}



clfile(fp_in, fp_out, fp_hist)

FILE *fp_in, *fp_out, *fp_hist;
{
                /* close the input and output files */
        fclose(fp_in);
```

```
        fclose(fp_hist);
}
```

```
/* Create a tree ( enter a new tree into the tree table ) */

/* USAGE:

  createe (module name) (parent tree name) (parent version number)
        (new tree name)

*/

#include (ctype.h)
#include (stdio.h)
#include "defines.h"
#include "tablestr.h"
#include "tabledef.h"
#include "filedefs.h"

#define error_0 "'createe'....\nRequired argument missing\n"
#define error_1 "'createe'....\n%s...module does not exist\n", modname
#define error_2 "'createe'....\n%s...Parent tree tree does not \
exist\n", argv[2]
#define error_3 "'createe'....\n%s...Invalid parent release.delta \
number\n", argv[3]
#define error_4 "'createe'....\n%s %s...Parent version does not \
exist\n", argv[2], argv[3]
#define error_5 "'createe'....\n%s...Tree already exists\n", argv[4]
#define error_6  "'createe' cannot open %s\n", tempfile
#define error_7 "'createe'....\n%s...New tree name does not begin with an \
alphanumeric charter\n", argv[4]
#define error_8 "'createe'....\n%s...New treename contains a reserved \
character\n", argv[4]

#define msg_1 "'createe'....Tree, %s, created.  It's parent is: %s \
%d.%d\n", argv[4], argv[2], prelse, pdelta


char reservedc[] = "*";  /* reserved characters */
        FILE *fp_mod, *fp_temp, *fopen();

main ( argc, argv )
int argc;
char *argv[];
{
        int prelse, pdelta;     /* parent release and delta numbers */
        char *treeargv; /* pointer to the tree name argument */
        char *chkrsvc();  /* check for a reserved character */
        struct treentry *getreptr();
        struct treentry *parnptr;  /* parent tree pointer */
        struct treentry *treeptr;  /* new tree pointer */
        struct relnode *verifyrd();
        char firstchr; /* first character in new tree name argument */
        char parnver[arg_size];  /* buffer for parent version number */
        char iobuff[buffsize];  /* i/o buffer */

                /* Make sure that all required arguments have been submitted*/
        if ( argc ( 5 )
        {       printf( error_0);
```

```
        exit (1);
}

        /* Open the TCS module file */
strcat( modname, argv[1] );
if ( (fp_mod = fopen( modname, "r" )) == NULL )
{
        printf(error_1);
        exit (1);
}

        /* Open the temporary new module file */
strcat( tempfile, argv[1] );
if ( (fp_temp = fopen( tempfile, "w" )) == NULL)
{
        printf( error_6 );
        exit (1);
}

        /* read the tree, release, and change tables and their
           sizes */
readtble( fp_mod, &tretblsz, &reltblsz, treetble, rtbnodes,
          &chgtbsz, chgtble );

        /* Check that the parent exists; get its pointer into
           the tree table */
if ( !(parptr = getreptr( argv[2], treetble )) )
{       printf( error_2);
        exit(1);
}

        /* Retreive the parent version number */
strcpy( parnver, argv[3] );

        /* Check the validity of the parent version number
           argument; Convert the release and delta number
           to integer form */
if ( !intrel( parnver, &prelse, &pdelta ) )
{       printf( error_3 );

        exit(1);
}

        /* Check the validity of the new tree name argument */
        /* A tree name must begin with a letter or a digit and
           cannot contain the wild card character, "*".  */
treeargv = argv[4];  /* get the pointer to the tree name argument */
firstchr = *treeargv; /* get the first character in the tree name */
if ( !(isalnum(firstchr)) )
{
        printf( error_7);
        exit(1);
}

        /* The tree name should not contain any reserved characters */
        /* Check the new tree name */
```

```
        if ( chkrsvc(argv[4], reservedc) )
        {
                printf( error_8);
                exit(1);
        }

                /* verify that the parent exists */
        if ( !verifyrd( parnptr, treetble, rtbnodes, prelse, pdelta) )
        {       printf( error_4 );
                exit(1);
        }

                /* Check that the new tree does not already exist */
        if( getreptr( argv[4], treetble ) )
        {       printf( error_5);
                exit(1);
        }

                /* Update the tree table by entering the new tree entry */
        if ( !( tretblsz = updtrtbl(argv[4], parnptr, prelse, pdelta,
                tretblsz, treetble)) )
                exit(1);

                /* Write the tables out to the new file */
        writabls( fp_temp, &tretblsz, &reltblsz, treetble, rtbnodes,
                  &chgtbsz, chgtble );

                /* Read and write out the body of the module file */
        rwbody( fp_mod, fp_temp, iobuff );

                /* Tell the user that the new tree has been created */
        printf( msg_1 );

                /* close files and exit */
        exit(0);
```

```
/* Install a 'd' (delete) type delta into the new module file */

#include (stdio.h)
#include "defines.h"
#include "contlstr.h"

char
*delete( from, to, contlptr, fp_diff, fp_temp, fp_out, diffbuff, iobuff )

        /*input arguments */
int from; /* first line to be deleted for the new version */
int to; /* last line to be deleted for the new version */
struct control *contlptr; /*pointer to the control record for the new version*/
FILE *fp_diff; /* diff file */
FILE *fp_temp; /* temporary module file with line numbered text lines */
        /* output arguments */
FILE *fp_out; /* temporary new module file */
char *diffbuff; /* contains a line from the diff file */
char *iobuff; /* contains lines from the temporary module file */
{
        int writincl; /* 'write including line' indicator */
        int lastline; /* keeps track of the last line number processed */
        int linenum; /* Buffer for text line number read in */
        struct control contlbuf; /* Buffer for reading control records */
        char *diffeof; /* end of diff file indicator */
        char  type;
                /* Read and write lines from the temporary file, up to but
                   not including 'from' line */
        writincl = 0;
        rwfile( from, writincl, fp_temp, fp_out, iobuff );

                /* Write the new 'delete' control character and record */
        type = delcontl;
        wrtcontl( type, contlptr, fp_out );

                /* Write the text line that 'rwfile' left in the
                   i/o buffer */
        putc( txtcontl, fp_out ); /* Write the text control character */
        fputs( iobuff, fp_out );

                /* Read and write up to and including the 'to' line */
                /* If 'from' = 'to', then only one line has been deleted
                   and was already written out.  Otherwise, there are more
                   lines to process */
        lastline = from;
        while ( lastline != to )
        {
                type = getc( fp_temp ); /* read a control character */
                switch (type) {
                        case delcontl:
                        case endcontl:
                                fread( &contlbuf, sizeof(struct control), 1,
                                        fp_temp);
                                wrtcontl( type, &contlbuf, fp_out);
                                break;
                        case inscontl:
```

```
                                /* write an end control for this
                                   version*/
                                wrtcontl( endcontl, contlptr, fp_out );
                                /* Read the insert control record */
                                fread( &contlbuf, sizeof(struct control),
                                1, fp_temp);
                                /* Write out the control record
                                   just read in. */
                                wrtcontl( inscontl, &contlbuf, fp_out );
                                /* Now start over as though this is
                                   is the beginning of the file */
                                lastline = lastline + 1;
                                rwfile( lastline, writincl, fp_temp,
                                fp_out, iobuff );
                                /* write the end control record for
                                this version */
                                wrtcontl( delcontl, contlptr, fp_out );
                                /* write the text record "rwfile"
                                   left in the i/o buffer */
                                putc( txtcontl, fp_out );
                                fputs( iobuff, fp_out );
                                break;
                        case txtcontl:
                                /* read in the line number */
                                fread( &linenum, sizeof(linenum), 1, fp_temp );
                                /* read the text */
                                fgets( iobuff, buffsize, fp_temp );
                                /* write the text control character */
                                putc( txtcontl, fp_out );
                                /* write the text record */
                                fputs( iobuff, fp_out );
                                /* If "linenum" = 0, then the text
                                   line read doesn't belong to the
                                   previous version.  Otherwise,
                                   set "lastline" to "linenum" */
                                if ( linenum != 0 )
                                        lastline = linenum;
                                break;

        }

        /* Write out the end control record */
        type = endcontl;
        wrtcontl( type, contlptr, fp_out);

        /* Ignore all diff lines that give the contents of the
           lines to be deleted */

        diffeof = fgets( diffbuff, difflnsz, fp_diff );
        while ( diffbuff[0] == '(' && diffeof != NULL )
                diffeof = fgets( diffbuff, difflnsz, fp_diff );

        return (diffeof);
```

```
/* Scan a diff line of the form:
        L1,L2t  where L1 is the first line affected, L2 is the last line,
                and t is the difference type: 'd' or 'c' for, delete
                or change respectively.
          OR

        0a    meaning to append before the first line,

          OR

        L1t   where if t = 'a', then append after L1, if t = 'd' then delete L1,
                or if t = 'c', then change L1.
*/
difflt( diffline, fromline, toline, difftype )

        /* input argument */
char *diffline;  /* line from diff file */
        /* output arguments */
int *fromline, *toline;  /* line range limits */
char *difftype; /* diff type: 'a', 'd', or 'c' */
{
        int endlinno;  /* end of line numbers found indicator */
        char *ptr;  /* traverses the line character by character */

        endlinno = 0;
        ptr = diffline + 1;
                /* Get the first line number */
        while ( !endlinno )
        {
                switch (*ptr){
                case ',':
                case 'a':
                case 'd':
                case 'c':
                                /* The end of the line number string was
                                   hit; convert the line number to integer
                                   form */
                        *fromline = *toline = convatob( diffline, ptr );
                        endlinno = 1;
                        break;
                default:
                        /* the end of the line number string hasn't been
                           hit yet; point to the next character */
                        ptr++;
                        break;
                }
        }

        if ( *ptr == ',' ) /* Is there a second line number? */
        {
                        /* point to the beginning of the second line
                           number string */
                diffline = ++ptr;
                ptr++;  /* Point to the second character in the substring */

                        /* Scan the substring until the end of the line
```

```
                    number has been hit ( 'ptr' points to a 'd'
                    or 'c').  */
            while ( !( *ptr == 'd'  || *ptr == 'c') )
                    ptr++;

                    /* convert the second line number to integer */
             *toline = convatob( diffline, ptr );
    }
    *difftype = *ptr; /* return the type character */
```

```
/* Get the pointer to the delimiter character in a string and check */
/* for any alpha characters in the string */
/* 'fndendch' returns a 0 if any alphas are found in the string */


char
*fndendch( strbegn, delimitr )

                /* input arguments */
char *strbegn; /* pointer to the beginning of the substring */
char delimitr;   /* character which delimits the substring    */

{
        char *dlmptr;   /* traverses the string */
        int numerch; /*indicates numeric character in the substring */

        /* search the string for the end character */


        numerch = 1;    /* assume the first character is numeric */
        dlmptr = strbegn;       /* point to the beginning of the string */

        do
        {       numerch = (*dlmptr )= '0' && *dlmptr (= '9' );
                dlmptr++;       /* point to the next character */
        } while ( (*dlmptr !=delimitr) && numerch);

        if ( numerch )
                return ( dlmptr );
        else
                return (0);
 }  /* end fndendch */
```

```
        /* get the tree's pointer and the release and delta numbers */
#include "defines.h"    /* constants definitions */
#include "tablestr.h"   /* table structures */

fndtrdnm ( treename, vername, rtbnodes, treetble, opt_indc,
         relnum, deltanum, treeptr, emptyptr, rdptr )

/* fndtrdnm returns false (0), if the version does not exist. */
/* Otherwise, it returns true (1) */

        /* input arguments */
char    *treename;      /* pointer to tree name */
char    *vername;       /* pointer to version name */
struct  relnode rtbnodes[];     /* release table nodes */
struct  treentry treetble[];    /* tree table */
int     opt_indc[];     /* optional arguments indicators */

        /* output arguments */
int *relnum;    /* release number pointer */
int *deltanum;  /* delta number pointer */
struct treentry **treeptr;      /* pointer to tree's entry in the tree table */
struct treentry **emptyptr;     /* tree pointer, if the tree is empty */
struct relnode **rdptr; /* ptr to release's entry in the release table */
{
        struct relnode *gethird();      /* release node pointer */
        struct treentry *getreptr();    /* tree pointer */
        struct relnode *versubm();      /* release node pointer */
        struct treentry *tmptptr;       /* temporary tree pointer */
        struct treentry *mainptr;       /* temporary main tree pointer */
        struct treentry *prntptr;       /* temporary parent tree pointer */
        int relindx;    /* release index */
        int tmprel;     /* temporary release number */
        int tmpdelta;   /* tempory delta number */
        int found;      /* error indicator */

        *emptyptr = false;      /* assume the tree is not empty */
        found = true;           /* assume numbers will be found */

        switch ( opt_indc[treearg] ) { /* tree number submitted indicator */
        case false:     /* no tree name submitted */
                        /* Return the pointer to the Main tree */
                        /* The Main tree's entry is at 'treetble[0]' */

                        *treeptr = mainptr = &treetble[0];

                /* The indicator for the version name argument is checked */
                /* If the version name (in the form of release.delta) was */
                /* not submitted, return the highest release and delta   */
                /* numbers from the main tree's release table. If it was */
                /* submitted, convert the ASCII form to integer, check that */
                /* the version does exist, and return the values submitted */

                switch ( opt_indc[versarg] ) {  /* version name arg. indicator*/
                case false:     /* no version name submitted */
```

```
                        /* get the highest release.delta numbers and */
                        /* and release pointer for the main tree */

                        *rdptr = gethird(mainptr, treetble, rtbnodes,
                                        &tmprel, &tmpdelta );
                        break;

                case true:      /* the version name was submitted */

                        /* Perform error checks on the version name */
                        /* If it's in valid form, 'found' will be set */
                        /* to 1, 'relnum' and 'deltanum' will contain */
                        /* the integer release and delta numbers, and   */
                        /* 'rdptr' contains the pointer to the release. */
                        /* otherwise, 'found' will contain a 0. */

                        if ( !(*rdptr = versubm( mainptr, vername, treetble,
                                rtbnodes, &tmprel, &tmpdelta) ) )
                                found = false;
                        break;
                } /* end case */
                break;
        case true:      /* tree name argument was submitted */
                /* get the pointer to the tree table entry */
                /* If the tree does not exist, 'getreptr' returns*/
                /* a 0, and an error message is printed here. */

                if ( *treeptr =  tmptptr = getreptr( treename, treetble ) )
                {       switch (opt_indc[versarg]) {
                        case false:     /* no version name argument submitted */
                                /* check for an empty tree */

                                if ( tmptptr->numrels == 0)
                                {       /* get the parent's release and */
                                        /* delta numbers. Get the parents' */
                                        /* tree pointer and set 'emptyptr*/
                                        /* to point to the tree requested */

                                        tmprel = tmptptr->prntrel;
                                        tmpdelta = tmptptr->prntdlta;
                                        prntptr = treetble + (tmptptr->ptblink);
                                        *treeptr = prntptr;
                                        *emptyptr = tmptptr;
                                        relindx = getrindx(prntptr, rtbnodes,
                                                tmptptr->prntrel);
                                        *rdptr = &rtbnodes[relindx];
                                }
                                else    /* get highest r.d */
                                        *rdptr = gethird(tmptptr, treetble,
                                                rtbnodes, &tmprel, &tmpdelta );
                                break;
                        case true: /* version name argument was submitted */
                                        /* Check for an empty tree */
                                if ( tmptptr->numrels == 0 )
                                {       printf("%s is an empty tree\n", treename);
                                        found = false;   /* indicate an error */
```

```
/* Get the index into the comment table for the requested version */

#include "defines.h"
#include "liststr.h"
#include "tablestr.h"

gcomindx( comtable, rtbnodes, rdptr, deltanum )

        /* input arguments */
struct cominfo comtable[];      /* Comment table */
struct relnode rtbnodes[];      /* release tables */
struct relnode *rdptr;  /* pointer to the release table entry */
int deltanum;   /* delta number of the requested version */

/* Function returns the index into the comment table */

{
        int index;      /* Used to traverse the comment table */

                /* Get the index of the first comment for this release */
        index = rdptr->firsthis;

                /* traverse the comment table list until the entry for
                   the delta is found */
        while ( --deltanum )
                index = comtable[index].nxtentry;

        return (index);
}
```

```
#include (stdio.h)
#include (time.h)
#include "defines.h"    /* constants */
#include "tablestr.h"   /* table structures */
#include "tabledef.h"   /* table definitions */
#include "filedefs.h"   /* filename array definitions */
FILE *fp_in, *fp_out, *fopen(), *fp_chg, *fp_hist;

                /* valid options for "GET" operation */
                /* 't' means a tree name is being submitted */
                /* 'v' means a version name, in the form of "release.delta"
                       number, is submitted */
                /* 'c' requests that the retreived version be opened for
                       change */
                /* 'p' requests the tree's path name */
                /* 'h' requests that the version's comment is printed */
char opt_arg[] = "tvcph";

        /* error messages */
#define error_3 "cannot open %s %d.%d for change \n...the last version can \
be oper....last version is %d.%d\n", treeptr->trename, relnum, \
deltanum, treeptr->numrels, rtbnodes[treeptr->lastrel].hipdelta

#define error_4  "%s is already open for change by %s\n", \
treeptr->trename, chgptr->userid

#define error_5 " cannot open module for change\n....cannot obtain user \
id for the change table entry\n"

main( argc, argv )

int argc;
char *argv[];

{
        int not_put;    /* 'get'called 'getversn' indicator */
        int opt_indc[num_opt];
        int argerror;   /* argument error indicator */
        int relnum, deltanum;   /* release and delta numbers */
        int linecnt;    /* line count */
        struct treentry *treeptr, *emptyptr;    /* tree pointers */
        struct relnode *rdptr;  /* release table pointer */
        char pathname[buffsize];        /* path name buffer */
        char treename[arg_size];
        char versname[arg_size];

                /* Open the TCS module file. The second argument should be
                   the module name */
        if( strlen(argv[1]) ) arg_size )  /* check module name size */
        {
                printf("%s...module name too long\n", argv[1]);
                exit (1);
        }
                /* get the TCS module name */
        strcat( modname, argv[1] );
                /* Open the file */
```

```
if ( (fp_in = fopen( modname, "r" )) == NULL )
{
        printf( "%s...TCS module does not exist\n", argv[1] );
        exit (1);
}

        /* Parce the optional argument list */
parcopta( argc, argv, opt_indc, opt_arg, &argerror, treename, versname);
if (argerror)   /* Was there an error in the argument list? */
        getout();       /* close files and exit */

        /* Read the tree , release, and change tables */
readtble( fp_in, &tretblsz, &reltblsz, treetble, rtbnodes,
        &chgtbsz, chgtble );

        /* Perform error checks on the arguments submitted.
           Get the tree pointers and the release and delta numbers */
if ( !fndtrdnm( treename, versname, rtbnodes, treetble, opt_indc,
                &relnum, &deltanum, &treeptr, &emptyptr, &rdptr ) )
        getout(); /* some error occurred in 'fndtrdnm' */

        /* If the file should be opened for change, then
           go into the change routine and update the module with
           the new change table. */
if ( opt_indc[changarg] )
{
                /* If the change request is ok, then the change
                   is entered in the change table, a new module
                   file is created and the tables are written out
                   here */
        if ( catlchg( treeptr, emptyptr, relnum, deltanum ) )
        {
                if ( (fp_chg = fopen( tmpfile2, "w")) == NULL )
                {       printf("cannot open %s\n", tmpfile2);
                        getout();
                }
                else
                {
                        chgtbsz++;  /*increment the change table size*/
                                /* write the tables to the new file*/
                        writabls( fp_chg, &tretblsz, &reltblsz,
                                treetble, rtbnodes, &chgtbsz, chgtble);
                }
        }
        else
                getout();
}
else
        fp_chg = 0;

        /* Build the internal tree structure */
bldintst( treeptr, relnum, deltanum, rtbnodes, treetble,
          intrestr, trestrtb );

        /* The retreived version goes to temporary file. Get
           the temporary file name and open the file */
```

```
strcat( tempfile, argv[1]);
if ( (fp_out = fopen( tempfile, "w")) == NULL )
{
        printf( "cannot open %s\n", tempfile );
        getout();
}

        /* Get the version requested and print the number of
           lines retreived */
        /* indicate that 'get' rather than 'put' is calling 'getversn'*/
not_put = 0;
linecnt = getversn( not_put, intrestr, trestrtb, fp_in, fp_out, fp_chg);
        /* Print the path name if it was requested */
if ( opt_indc[pathnarg] )
{
                /* First print the module name */
        printf("%s\n", argv[1]);
                /* If the tree is empty, set the release and delta
                   number to 0 and send 'gpthname' the empty tree's
                   pointer */
        if ( emptyptr )
                gpthname( emptyptr, treetble, 0, 0, pathname );
        else
                gpthname( treeptr, treetble, relnum, deltanum,
                        pathname);
                /* Print the pathname */
        printf( "%s\n", pathname );
}
else
{
        /* The path name was not requested, just print this
           version's name */
        if ( emptyptr )
        {
                printf( "%s, %s\n", argv[1], emptyptr->trename );
                printf("Tree is empty.\nParent retrieved: %s %d.%d\n",
                        treeptr->trename, relnum, deltanum );
        }
        else
                printf("%s %s %d.%d\n", argv[1], treeptr->trename,
                        relnum, deltanum );
}

printf("%d lines retreived\n", linecnt);

        /* If the user also wants the comment for this version,
           print it. */
if ( opt_indc[histarg] )
{
                /* Open the history file */
        strcat( histfile, argv[1]);
        if ( (fp_hist = fopen( histfile, "r")) == NULL )
        {
                printf("Cannot open %s\n", histfile);
                printf("cannot obtain comment for version\n");
        }
```

```
                if ( !gethis( rtbnodes, rdptr, fp_hist, deltanum ) )
                        printf("Cannot obtain comment\n");
        }
        exit (0);
}    /* end get */


catlchg( treeptr, emptyptr, relnum, deltanum )
        /* place a new entry into the change table */
        /* The entry is placed at the end of the table */
struct treentry *treeptr, *emptyptr;
int relnum, deltanum;
{
        struct treentry *tmptr;
        struct change *chgptr, *chkopen();
        char *idptr, *getlogin();
        long time();

                /* If the tree is empty, 'emptyptr' contains its pointer */
        if ( emptyptr )
                treeptr = emptyptr;
        else
                        /* Check that the last version is to be changed */
                if ( !(treeptr->numrels == relnum &&
                        rtbnodes[treeptr->lastrel].hipdelta == deltanum) )
                {
                        printf( error_3 );
                        return (0);
                };

                /* Check the change table for the tree number entry. If
                   it's in the table, the version is already open for change*/
        if ( chgptr = chkopen( treeptr, chgtble, chgtbsz ) )
        {
                printf( error_4);
                return (0);
        }

        chgptr = chgtble + chgtbsz; /* point to the new entry */
                /* Put the info into the change table */
        chgptr->trenum = treeptr->trenum; /* enter the tree number */
                /* Get the user's ID and enter it into the table */

        if ( (idptr = getlogin() ) == NULL )
        {       printf(error_5);
                return (0);
        }
        strcpy( chgptr->userid, idptr );



                /* Enter the time and date info */
        time( &chgptr->time );

        return (1);
} /* end catlchg */
```

```
getout()
        /* This function is called if some error */
{
        fclose(fp_in);
        exit (1);
}
```

```
/* Read the comment from the standard input. A null line marks the end of
   the comment */

#include (stdio.h)

#define cprompt "Enter comment...\n@ to %d characters\n\n", buffsize-1
getcommt( iobuff, buffsize )

char *iobuff;   /* i/o buffer */
int buffsize;   /* i/o buffer size */

{
        int linesz;     /* number of character in a line read in */
                /* maximum line size; number of characters left in 'iobuff' */
        int limit;
        int numchar;    /* total number of characters in comment */
        int end;        /* end of comment indicator */

        limit = buffsize - 1;   /* make sure the buffer has room for the '\0'*/
        end = !EOF;
        linesz = 0;
        numchar = 0;    /* set the number of characters read */

                /* Prompt for the comment */
        printf( cprompt );

        while ( (limit -= linesz) && end != EOF )
        {
                        /* Prompt for a line. The prompt is the number of
                           characters read so far */
                printf( "%d: ", numchar );
                        /* read a line from the standard input and get its
                           size in characters */
                linesz = getline( iobuff, limit, &end  );
                numchar += linesz;      /* count the characters in the line*/
                iobuff += linesz ; /* point to the next buffer position */
        }
                /* Mark the end of the comment with a '\0' */
        *iobuff = '\0';

                /* Let the user know the comment size entered */
        printf( "\nComment entered....%d characters\n", numchar );
                /* return the total comment size including the '\0' */
        return (numchar + 1);
}  /* end getcommt */


getline( iobuff, lim, eof )

char *iobuff;
int lim;        /* maximum line size */
int *eof;       /* end of file indicator */

/* The function returns the number of characters read */
{
        int charcnt;    /* character counter */
```

```
        int c;  /*character buffer */

        charcnt = 0;    /* initalize the character count */

                        /* Increment the character count; get a character. */
        while ( charcnt < lim && (c = getchar()) != EOF  && c != '\n' )
        {
                        /* Put the character into the buffer and point to the
                           next available buffer position */
                *iobuff++ = c;
                charcnt++;      /* count the character */
        }

        if ( c = '\n' );  /* Was the last character the newline? */
        {
                if ( charcnt == 1 && *(iobuff-1) == '.' )
{
                        charcnt = 0;
                        *eof = EOF; /* signal end of comment */
                }
                else
                {
                        *iobuff = c;
                        charcnt++; /* count the character */
                }
        }

        return (charcnt);
}
```

```
/* Get the pointer to the highest release in the tree. Also return the */
/* highest release and delta numbers */

#include "defines.h"    /* program constants */
#include "tablestr.h"   /* table structures */

struct relnode
*gethird( treeptr, treetble, rtbnodes, relnum, deltanum )

        /* input arguments */
struct treentry *treeptr;       /* tree pointer */
struct treentry treetble[];     /* tree table */
struct relnode rtbnodes[];      /* release table */
        /* output arguments */
int *relnum;    /* release number */
int *deltanum;  /* delta number */

        /* The function returns the pointer to the last release */

{
        struct relnode *rdptr;  /* temporary release pointer */

                /* return the highest release from the tree table */
        *relnum = treeptr->numrels;
                /* get the pointer to the last release's node in the /
                /* release table */
        rdptr = &rtbnodes[treeptr->lastrel];

                /* return the highest propagating delta number */
        *deltanum = rdptr->hipdelta;
        return (rdptr);
}
```

```
/*  Open the history file and print the comment for the requested version */

#include (stdio.h)
#include (time.h)
#include "defines.h"
#include "histstr.h"
#include "histdefs.h"
#include "tablestr.h"

gethis( rtbnodes, rdptr, fp, deltanum )

        /* input arguments */

struct relnode rtbnodes[];      /* Release tables */
struct relnode *rdptr; /* Pointer to the release table entry for this version*/
FILE *fp;       /* pointer to the history file */
int deltanum;   /* delta number requested */

/* Function returns a 0 if some error occurred.  Otherwise it returns 1. */

{
        int cindex;     /* index into the comment table */
        long offset;    /* offset from the beginning of the history body */
        char *ctime();  /* Time conversion function */
        char comment[buffsize]; /* Buffer for comment read in */


                /* Read the history size and the comment table size */
        fread( &hstsizes, sizeof(struct hsizes), 1, fp );

                /* Read the comment table */
        fread( comtable, sizeof(struct cominfo), hstsizes.comtbsz, fp );

                /* Get the index into the comment table for this version */
        cindex = gcomindx( comtable, rtbnodes, rdptr, deltanum );

                /* Get the offset for the comment block */
        offset = comtable[cindex].offset;

                /* Position the file pointer at the offset */
        fseek( fp, offset, 1 );

                /* Read in the comment */
        fread( comment, sizeof(*comment), comtable[cindex].blksize, fp );

                /* First print the change info: by whom and when */
        printf("Created by %s, on %s\n", comtable[cindex].userid,
                ctime( &comtable[cindex].when ) );
                /* Print the comment */
        printf( "%s\n", comment );

        return (1);
}
```

```
/* get the pointer into the tree table */

#include "defines.h"    /* constants */
#include "tablestr.h"

struct treentry
*getreptr( tree_in, treetble )

        /* search the tree table for 'tree_in'. */

        /* The function returns the pointer into the tree table, if the */
        /* entry is found. Otherwise it returns a 0. */

        /* input arguments */
char *tree_in;  /* tree name submitted by the user */
struct treentry *treetble;      /* tree table address */
{
        for (  ; treetble->trename[0];  treetble++ )
                if ( strcmp( tree_in, treetble->trename ) == 0 )
                        return ( treetble );  /* return the tree pointer */
        return (0);
} /* end getreptr */
```

```
        /* All pointer have been set up; get the version requested */
#include (stdio.h)
#include "defines.h"    /* constants */
#include "tablestr.h"   /* table structure definitions */
#include "contlstr.h"   /* control structures and constants */

#define error_1  ("Error reading TCS file: no text follows control character '%c'\n", type )

#define error_2 (" Invalid control character '%c' encountered\n", type )


int linecnt;    /* line counter */
int flag;       /* flag indicating whether or not a delta is applied*/
int takeline;   /* indicates whether or not a line of text is taken */
char iobuff[buffsize];  /* i/o buffer */
char type;      /* control record type */
struct control contlrec;        /* control record */
int is_ok;      /* indicates: no error in creating the TCS file */

getversn( put, intrestr, trestrtb, fp_in, fp_out, fp_chg )

        /* input arguments */
int put; /* if positive, indciates that 'put' called this function */
int intrestr[]; /* internal tree structure */
struct refs trestrtb[]; /* cross reference table for internal tree structure */
FILE *fp_in;  /* module input file */
        /* output arguments */
FILE *fp_out; /* file containing the version delivered to the user */
        /* If 'get' called this function 'fp_chg' will contain a file pointer
           if the user is opening the module for change purposes.  In this
           case, 'getversn' will write out to the new module file, pointed to
           by 'fp_chg', the body of the module.

           If 'put' called this function then 'fp_chg' will contain the pointer
           to the temporary line numbered file.  'getversn then writes out
           all control info and line numbered text records to this file */
FILE *fp_chg;

        /*function output */
/* The function returns the line count if the version was retreived; otherwise
   it returns 0. */

{
        initfree();     /* initiialize the free list */
        initque();      /* initialize the control queue */
        linecnt = 0;    /* initialize the line counter */
        is_ok = 1;      /* assume the version will be retrieved */

                /* Read the the file line by line and write out those lines
                   belonging to the version requested */
        while ( (type = getc(fp_in)) != EOF ) /* read the control character */
        {
                if (fp_chg)
                                /* Write the control character to the tempoarary
                        putc( type, fp_chg);
                switch (type){  /* control or text record? */
```

```
                case inscontl:
                case delcontl:
                        /* an insert or delete control record follows
                           the control character */

                                /* process the control record */
                        i_d_rec( intrestr, trestrtb, fp_in, fp_chg);
                        break;
                case endcontl:
                                /* An end control record follows the
                                   control character */
                                /* process the end control record */
                        end_rec( fp_in, fp_chg );
                        break;

                case txtcontl:
                                /* a line of text follows control char. */

                                /* process a line of text */
                        textline( out, fp_in, fp_out, fp_chg );
                        break;
                default:
                        /* illegal control character encountered: an error*/
                        printf( error_2 );
                        is_ok = 0;
                        break;
                }
                        /* If an error occurred during file processing,
                           return a 0 to indicate the error. */
                if ( !is_ok )
                        return (0);
        } /* end while */

        return ( linecnt );     /* return the line count */
}


i_d_rec( intrestr, trestrtb, fp_in, fp_chg )
        /* process an insert or delete control record */

int intrestr[]; /* internal tree structure */
struct refs trestrtb[]; /* cross reference table for internal tree structure */
FILE *fp_in;    /* input file pointer */
FILE *fp_chg; /* If positive, contains the output file pointer */

{
                /* read the control record */
        if ( fread(&contlrec, sizeof(struct control), 1, fp_in) )
        {
                if( fp_chg)
                        fwrite(&contlrec, sizeof(struct control), 1, fp_chg);
                        /* set the flag */
                flag = setflag(&contlrec, intrestr, trestrtb,
                                type);
```

```
                        numbers, and the flag into the
                        control queue */
                if ( enterque(&contlrec, flag) )
                                /* indicate whether or not
                                   the next line of text
                                   should be taken */
                        takeline = setkline(flag);
                else
                        /* the queue was full */
                        is_ok = 0;
        }
        else
                /* for some reason, no record was read */
        {
                printf( error_1 );
                is_ok = 0;
        }
}


end_rec( fp_in, fp_chg )
        /* process an end control record */
FILE *fp_in;    /* input file pointer */

FILE *fp_chg;  /* If positive, contains the output file pointer */
{
                /* read the control record */
        if( fread(&contlrec, sizeof(struct control), 1, fp_in) )
        {
                if ( fp_chg )
                        fwrite( &contlrec, sizeof(struct control), 1, fp_chg);
                        /* remove an entry from the queue */
                leaveque(&contlrec);
                        /* get the flag field of the
                           entry at the head of the
                           queue */
                flag = headflag();
                        /* reset the "take line"
                           indicator */
                takeline = setkline( flag );
        }
        else
        {
                /* no record was read following the control
                   character: an error */
                printf( error_1 );
                is_ok = 0;
        }
}


textline( put, fp_in, fp_out, fp_chg )
        /* process a line of text */
int put; /* If positive, then 'put' called 'getversn' */
FILE *fp_in;  /* input file pointer */
        /* output arguments */
```

```
FILE *fp_out;  /* output file */
FILE *fp_chg; /* If positive contains the temporary new module file pointer*/

{
        int lineno; /* line number */
                /* read a line of text. If there isn't one, an error occurred*/
        if ( fgets( iobuff, buffsize, fp_in) == NULL )
        {
                printf( error_1 );
                is_ok = 0;
        }
        else
        {
                /* If 'put' called 'getversn', assign a line number to the
                   line and write out the line number and the line to the
                   'out' file */

                if ( put )
                {
                        if ( takeline == yesflag )
                                lineno = linecnt + 1;
                        else
                                lineno = 0;
                        fwrite( &lineno, sizeof(lineno), 1, fp_chg);
                }

                if ( fp_chg )
                        fputs( iobuff, fp_chg );

                /* If the line should be taken, write it
                out to the user's file; otherwise, ignore
                the line */

                if (takeline == yesflag)
                {
                        fputs( iobuff, fp_out );
                        linecnt++;      /* count the line*/
                }
        }
}
```

```
/* If the optional parameter submitted was a 't' or 'v', extract the
   tree or version name from the argument list */

#include "defines.h"


gopt( argc,  argv, option, tre_name, ver_name)

        /* input arguments */
int argc;
char *argv[];
char option;    /* option letter submitted */
        /* output arguments */
char *tre_name;   /* tree name argument returned if present */
char *ver_name;   /* version name argument returned if present */

/* The function returns a 1 if an error in the argument list was
    encountered.  Otherwise it returns 0. */


{

                /* extract the tree name or */
                /* version name if submitted */
        switch (option) {
        case 't' :
                /* If another argument follows the "-t", assume its
                   a tree name.  Send it back to the caller via "tre_name" */
                /* If no argument follows, indicate the error and return */
                /* Also check to see that there's room in the "tre_name array*/

                if ( argc ) 1 )
                {
                    if(strlen(argv[0]) ) arg_size)
                    {   printf("tree name too long");
                        return (1);
                    }
                    else
                        copy(tre_name, argv[0]);
                }
                else
                {
                        printf( "Tree name missing\n");
                        return (1);
                }
                break;
         case 'v' :
                /* If another argument follows the "-v", assume its
                   a version name.  Send it back to the caller via
                   "ver_name */
                /* If no argument follows, indicate the error and return */
                /* Also check to see that there's room in the "ver_name array*/

                if ( argc ) 1 )
                {
                    if(strlen(argv[0]) ) arg_size)
```

```
                {   printf("version name too long");
                    return (1);
                }
                else
                    copy(ver_name,argv[0] );
            }
            else
            {
                    printf( "version name missing\n");
                    return (1);
            }
            break;
        }

        return (0);
}
```

```
/* Get the pathname name string for the requested version */

#include (stdio.h)
#include "defines.h"
#include "tablestr.h"

struct nameinfo{
        char versname[ 2 * arg_size + 5 ];
        int versnsz;
}

#define maxperln 70

gpthname( treeptr, treetble, relnum, deltanum, pathname )

        /* Input arguments */
struct treentry *treeptr;       /* pointer into the tree table */
struct treentry treetble[];     /* Tree table */
int relnum;     /* release number of requested version */
int deltanum;   /* deltanumber of the requested version */
        /* Output arguments */
        /* Path name string returned with a '\0' delimiter */
char *pathname;

{
                /* Table used in building the path name */
        struct nameinfo tmptable[maxtrees];
        struct nameinfo *tmptr; /* Used to traverse "tmptable" */
        int pindex;     /* index of parent entry in the tree table */
        int tindex;     /* used to traverse the tree table */
        int charcnt;    /* counts characters on a line */

        tmptr = tmptable; /* point to the beginning of the temporary table*/
                /* Put the tree name and release.delta number for the
                   requested version into the temporary table */
        sprintf( tmptr->versname, "%s %d.%d",
                treeptr->trename, relnum, deltanum );

        tmptr->versnsz = strlen(tmptr->versname);
                /* Now convert the tree pointer to an index */
        tindex = treeptr - treetble;

                /* Put the parent's version name info into the table,
                   for each tree in the path */
      while ( ( pindex = treetble[tindex].ptblink ) != 0 )
      {
                tmptr++;  /* get the address of the next temp table entry*/
                        /* Put the parent's tree name, release.delta number,
                           and version name size into the temp table */
                sprintf( tmptr->versname, "%s %d.%d/ ",
                        treetble[pindex].trename, treetble[tindex].prntrel,
                        treetble[tindex].prntdlta );
      tmptr->versnsz = strlen(tmptr->versname);
                        /* Point to the parent tree's table entry */
                tindex = treetble[tindex].ptblink;
      }
```

```
            /* Now traverse the table backwards and format the path
               name array, putting in new line characters.  If the
               maximum number of characters per line would be exceeded
               by a name entry, that entry is put on the next line;
               i.e. names are not split between lines. */

        charcnt = 0; /* Initialize the character per line counter */
        while ( tmptr >= tmptable )
        {
                if ( (charcnt + tmptr->versnsz) > maxperln )
                {
                        /* The character per line limit would be
                           exceeded.  Put in the new line character and
                           re-initalize the count */

                        *pathname = '\n';
                        pathname++;  /* point to the next position */
                        charcnt = 0;
                 }

                        /* Put the version name into the path name */
                strcpy( pathname, tmptr->versname );

                        /* Get the address of the next available
                           position in "pathname."  This will cause the
                           '\0' from the last name to be overwritten
                           with the next name.  */
                pathname = pathname + tmptr->versnsz;
                charcnt += tmptr->versnsz;      /* count the characters*/
                tmptr--;        /* point to the next name */
          }
}
```

```
/* convert the version name argument string to release and delta number */
/* integers. The function returns a 1 if the conversion was successful  */
/* or a 0 if it wasn't */

#include "defines.h"

intrd( versname, relnum, deltanm )

        /* input arguments */
char *versname;
        /* output arguments */
int *relnum;
int *deltanm;

{
        char *rdlmptr; /* pointer to the release delimitor character */
        char *fndendch();       /* pointer to a string delimiter */
        char *dldlmptr; /* pointer to the delta delimiter */
        char *delsubst; /* delta substring */

        /* find the end of the release substring; the beginning is the */
        /* first character of versname; reldelim contains the delimiter */

        if ( rdlmptr = fndendch( versname, reldelim ))
        {
                /* Find the end of the delta string. The beginning is one */
                /* character beyond the release delimiter character. The */
                /* delta delimiter is the null character. */

                delsubst = rdlmptr + 1;         /* point to the delta beginning */
                if ( dldlmptr = fndendch ( delsubst, nullchar) )
                {
                        /* convert the release and delta numbers from */
                        /* ASCII to binary integers */

                        *relnum = convatob( versname, rdlmptr );
                        *deltanm = convatob( delsubst, dldlmptr);
                        return (1);     /* indicate successful conversion */

                }
        }
        return (0);     /* indicate unsuccessful conversion */
}
```

```
/* open all files used by 'put' */

#include (stdio.h)
#include "defines.h"
#include "filedefs.h"

openall( module, fp_mod, fp_old, fp_temp, fp_hist, fp_htmp )
        /* open the input and output files */

        /* input argument */
char *module;   /* module name */
        /* output arguments */
FILE **fp_mod;  /* TCS module file */
FILE **fp_old;  /* file containing the old version to be used by 'diff'*/
FILE **fp_temp; /* file containing the line numbered text to be used
                   by 'putdelta' */
FILE **fp_hist; /* history file */
FILE **fp_htmp; /* temporary new history file */

/* The function returns a 1 if all files were successfully opened. If any
   'open' fails the function returns a 0. */

{
        FILE *fopen();

                /* get the tcs module name and open the module file */
        strcat( modname, module );
        if ( (*fp_mod = fopen(modname, "r")) == NULL )
        {       printf("%s...module does not exist\n", modname );
                return (0);
        }

                /* The next file will have the old version for 'diff' */
        strcat( tempfile, module );     /* get the file's path name */
        if ( (*fp_old = fopen(tempfile, "w")) == NULL )
        {       printf("'put' cannot open %s\n", tempfile);
                return (0);
        }

                /* The next file will have the new tables and the body
                   of the module will have text lines numbers. This file
                   is used by 'putdelta' */
        if( (*fp_temp = fopen( tmpfile2, "w")) == NULL )
        {       printf("'put' cannot open %s\n", tmpfile2);
                return (0);
        }

                /* open the history file */
        strcat( histfile, module);  /* get the file's full path name */
        if ( (*fp_hist = fopen( histfile, "r")) == NULL )
        {       printf( "%s...history file does not exist\n", histfile);
                return (0);
        }

                /* open the temporary new history file */
        if ( (*fp_htmp = fopen( histmp, "w")) == NULL )
```

```
        {       printf("'put' cannot open %s\n", histmp );
                return (0);
        }

        return (1);
} /* end openall */
```

```
#include "defines.h"


parcopta(argc, argv, opt_indc, opt_arg, arg_err, tre_name, ver_name)
     /* parce the optional argument string submitted*/

int argc;
char *argv[];
int opt_indc[];    /* optional arguments indicators */
char opt_arg[]; /* valid options */
int *arg_err;      /* pointer to argument error indicator */
char *tre_name;   /* tree name argument returned if present */
char *ver_name;   /* version name argument returned if present */


{
        /* scan the argument list for optional arguments, which start */
        /* with the third argument. an option begins with a "-" and the */
        /* second character is one which appears in the optional args. */
        /* array, "opt_args[]".*/

        /* if an optional arg. is found its corresponding indicator in */
        /* array "opt_indc[]" . Extract the tree name and the version name */
        /* if either or both arguments were submitted */


char *list;   /* used to traverse the valid options array */
int *addr;   /* used to traverse the optional indicators array */
int i;       /* subscript into the optional arguments indicators */
int numopta = num_opt;     /* number of valid options */


              /* initialize the argument error indicator */
        *arg_err = 0;
              /* initial all optional argument indicators to 0 */
        for (addr = opt_indc; --numopta ) 0; *addr++ = 0 );
              /* point to the address of the second argument */
        argv++;

        while ( --argc)1 && *arg_err == 0 )
        {       /* point to the next argument address */
                /* the first character should be a "-" */

                if ( (*++argv)[0] != '-' )
                {       printf("%s  option specification incorrect \n", *argv);
                        printf(" preceed an option with a '-'\n");
                        *arg_err = true; /* flag the error */
                }
                else
                {       i = 0;  /* initialize the index into the indicators*/
                        /* the second character should be listed in opt-args*/
                        for ( list = opt_arg; *list != '\0' &&
                                *list != (*argv)[1]; list++)
                             /* increment the indicator's index*/
                             i++;
```

```
if (*list == '\0')
        /* the end of the valid options list was hit */
{       printf("%s  invalid option\n", *argv );
        *arg_err = true;  /* flag the error */
}
else
{ if (opt_indc[i])  /* check for duplicates */
  {     printf("%c   duplicate option submitted\n",
                (*argv)[1]);
         *arg_err = true; /* flag the error */
  }
  else
  {     opt_indc[i] = true;

                /* extract the tree name or */
                /* version name if submitted */
                switch (*list) {
                case 't' :
                case 'v' :
                        /* point to the next arg. */
                        argv++;
                        /* count the argument */
                        argc--;

                        *arg_err = goat( argc, argv,
                        *list,tre_name,ver_name);
                        break;
                 }
```

```
/* print the tree and release tables */
#include (time.h)
#include "defines.h"
#include "tablestr.h"

#define table_1 "%s\t%d\t%d\t%d\t%d\n", t->trename, \
t->trenum, t->ptblink, t->prntrel, t->prntdlta

#define table_2 "\t%d\t%d\t%d\t%d\n\n", t->numrels, \
t->firstrel, t->lastrel, t->newrflag

#define rtable "%d\t%d\t%d\t%d\n", r->hipdelta, \
r->nxtrnode, r->firsthis, r->lasthis

#define ctable "%s%c\t%s\n", ctime(&c->time), c->trenum, c->userid

printbl( tsize, rsize, treetble, reltable, csize, chgtble )
int tsize, rsize;       /* tree and release table sizes */
struct treentry treetble[];
struct relnode reltable[];
int csize;      /* change table size */
struct change chgtble[];        /* change table */
{
        struct treentry *t;
        struct relnode *r;
        struct change *c;
        char *ctime();

        printf("tree table size = %d \n", tsize );

        t = treetble;
        while ( t < treetble + tsize )
        {
                printf ( table_1 );
                printf ( table_2 );
                t++;
        }       /*end while */

        printf("\n");

        printf("release tables size = %d \n", rsize );
        r = reltable;
        while ( r < reltable + rsize )
        {
                printf( rtable );
                r++;
        }       /*end while*/

        printf("\n");
                /* print the change table */
        printf( "change table size = %d\n", csize);
        c = chgtble;
        while ( c < chgtble + csize )
        {
                printf( ctable );
                c++;
```

```
        }   /* end while */
        printf("\n");

}
```

```
/* Print out the TCS module file

       The Command line:   printemp (module name)
*/

#include (stdio.h)
#include "defines.h"    /* constants definitions */
#include "contlstr.h"   /* control record structure and definition */
#include "tablestr.h"
#include "tabledef.h"
#include "idinfo.h"
FILE *fopen(), *fp_in, *fp_out;
char iobuff[buffsize];  /* i/o buffer */
char type;      /* control character type */
struct control contlrec;        /* control record */

#define error_1  ("no values follow control character...%c\n", type)
#define error_2  ("no string follows control character...%c\n", type)

#define pcontl "%d %d.%d\n", contlrec.treenum, contlrec.relnum, contlrec.deltanum
main( argc, argv )

int argc;       /* number of arguments */
char *argv[];   /* argument pointers */

{
       int lineno;     /* line number buffer */


              /* make sure that a module name was submitted */
       if (argc < 2 )
       {
              printf( "module name missing\n" );
              exit (1);
       }


              /* concatenate the module name with the directory
              name for TCS */

              /* open the input file */
       if ( ( fp_in = fopen( argv[1], "r") ) == NULL )
       {
              printf( "cannot open %s\n", argv[1] );
              exit (1);
       }
              /* Read the id info */
       fgets( idbuff, idbufsiz, fp_in );
       printf( "%s\n", idbuff );
              /* Read and Write the control record */
       fread( &contlrec, sizeof( struct control), 1, fp_in );
       printf( pcontl);

              /* read the tree, release, and change tables */
       readtble( fp_in, &tretblsz, &reltblsz, treetble, rtbnodes,
              &chgtbsz, chgtble );
```

```
        /* print the tables */
    printbl( tretblsz, reltblsz, treetble, rtbnodes, chgtbsz, chgtble );

        /* print out the lines in the tcs module */
        /* If the type is a control record, print out the control
        character and the tree, release and delta numbers. If the
        line is a text record, just print out the text */

                /* read the control character */
    while ( ( type = getc(fp_in) ) != EOF )
    {
        switch (type){
        case inscontl:
        case endcontl:
        case delcontl:
                if( fread(&contlrec, sizeof(struct control),
                        1, fp_in) )
                        printf( "%c %d %d.%d\n", type,
                        contlrec.treenum,
                        contlrec.relnum,
                        contlrec.deltanum );
                else
                        printf( error_1 );
                break;
        case txtcontl:
                        /* print the control character */
                printf("%c", type);
                        /* Read and print the line number */
                fread( &lineno, sizeof(lineno), 1, fp_in);
                printf(" %d ", lineno );
                if ( fgets( iobuff, buffsize, fp_in) == NULL)
                        printf( error_2 );
                else
                        printf(" %s",  iobuff);
                break;
        default:
                printf("invalid control character...%c  ",
                        type);
                break;
        }
    }
    fclose(fp_in);
}
```

```
#include (time.h)
#include (stdio.h)
#include "defines.h"
#include "histstr.h"
#include "histdefs.h"

main( argc, argv )

int argc;
char *argv[];

{
        FILE *fp, *fopen();
        struct cominfo *ptr;    /* used to traverse the comment table */
        struct cominfo *limit;  /* pointer to the last comment table entry */
        int numentry;
        char comment[buffsize];
        char *ctime();

                /* Open the history file */
        if ( (fp = fopen( argv[1], "r")) == NULL )
        {       printf( "'printhis' cannot open %s\n", argv[1]);
                exit (1);
        }

                /* read the comment table and history size */
        fread( &hstsizes, sizeof(struct hsizes), 1, fp );

                /* Print out the sizes */
        printf("comment table size = %d\n", hstsizes.comtbsz);
        printf("history size = %d\n", hstsizes.histsz);
                /* read the comment table */
        fread( comtable, sizeof(struct cominfo), hstsizes.comtbsz, fp );

                /* print out each comment */
        numentry = hstsizes.comtbsz;
        for ( ptr = comtable, limit = comtable + numentry; ptr ( limit; ptr++ ) {
                        /* Print out the comment table entry */
                printf( "\nwhen..%s", ctime(&ptr->when) );
                printf( "who..%s\t", ptr->userid );
                printf( "block size = %d\t", ptr->blksize);
                printf( "offset = %d\t", ptr->offset);
                printf( "next entry index = %d\n", ptr->nxtentry);
                        /* Read in the comment */
                fread( comment, sizeof(*comment), ptr->blksize, fp );
                        /* print it out */
                printf("%s\n", comment);
        }
}
```

```
/* Print out the TCS module file

       The Command line:   printmod (module name)
*/

#include (stdio.h)
#include "defines.h"    /* constants definitions */
#include "contlstr.h"   /* control record structure and definition */
#include "tablestr.h"
#include "tabledef.h"
#include"filedefs.h"
FILE *fopen(), *fp_in, *fp_out;
char iobuff[buffsize];  /* i/o buffer */
char type;      /* control character type */
struct control contlrec;        /* control record */

#define error_1  ("no values follow control character...%c\n", type)
#define error_2  ("no string follows control character...%c\n", type)
main( argc, argv )

int argc;       /* number of arguments */
char *argv[];   /* argument pointers */

{

        /* make sure that a module name was submitted */
    if (argc < 2 )
    {
        printf( "module name missing\n" );
        exit (1);
    }



        /* open the input file */
    if ( ( fp_in = fopen( argv[1], "r") ) == NULL )
    {
        printf( "cannot open %s\n", argv[1] );
        exit (1);
    }

        /* read the tree, release, and change tables */
    readtble( fp_in, &tretblsz, &reltblsz, treetble, rtbnodes,
        &chgtbsz, chgtble );
        /* print the tables */
    printbl( tretblsz, reltblsz, treetble, rtbnodes, chgtbsz, chgtble );

        /* print out the lines in the tcs module */
        /* If the type is a control record, print out the control
        character and the tree, release and delta numbers. If the
        line is a text record, just print out the text */

            /* read the control character */
    while ( ( type = getc(fp_in) ) != EOF )
    {
        switch (type){
```

```
                case inscontl:
                case delcontl:
                case endcontl:
                        if( fread(&contlrec, sizeof(struct control),
                                  1, fp_in) )
                                printf( "%c %d %d.%d\n", type,
                                contlrec.treenum,
                                contlrec.relnum,
                                contlrec.deltanum );
                        else
                                printf( error_1 );
                        break;
                case txtcontl:
                        if ( fgets( iobuff, buffsize, fp_in) == NULL)
                                printf( error_2 );
                        else
                                printf("%c %s", type, iobuff);
                        break;
                default:
                        printf("printmod: invalid control character...%c",
                                type);
                        break;
                }
        }
        fclose(fp_in);
}
```

```
/* show the highest delta in each release for a tree requested by the
   "SHOW TREE" command.  See 'show.doc' for documentation */

#include "defines.h"
#include "tablestr.h"
#include "histstr.h"
#include "showextn.h"   /* external table definitions */

prtrel( treeptr )

struct treentry *treeptr;       /* pointer to tree's entry in tree table */

{
        int release;
        int cnt;
        int relindx;

                /* Show the number of releases */
        printf( "Number of releases: %d\n", treeptr->numrels );
                /* If the tree is empty, say so and return */
        if ( !treeptr->numrels )
        {
                printf( "Tree is empty\n" );
                return (0);
        }

                /* Show the highest version in each release in the form
                   of r.d.  Versions are printed three to a line */
        release = 1;
        relindx = treeptr->firstrel;
        cnt = 0;

        while ( release <= treeptr->numrels )
        {
                if ( cnt == 3 )
                {
                        printf("\n");
                        cnt = 0;
                }

                printf( "%d.%d\t", release, rtbnodes[relindx].hipdelta );
                relindx = rtbnodes[relindx].nxtrnode;
                release++;
                cnt++;
        }

        printf("\n");
        return (0);
}
```

```
/* Print the information that the user requested in the 'SHOW TREE'
   command.  See 'show.doc' for documentation */

#include (stdio.h)
#include (time.h)
#include "defines.h"
#include "tablestr.h"
#include "histstr.h"
#include "showextn.h"   /* external table definitions */
#include "shotrdef.h"

prtrinfo( treeptr, showind, fp, fp_hist )

        /* input arguments */
struct treentry *treeptr;       /* pointer to tree table entry */
int showind[];  /* options indicators */
FILE *fp, *fp_hist;     /* TCS module and history files */

{
        int relindx;    /* index into the release table */
        int comindx;    /* index into the comment table */
        struct change *chgptr, *chkopen();  /* change table pointers */
        char *ctime();
        char pathname[buffsize];        /* path name string */

                /* First print the tree's name */
                /* If the -p options was submitted, print the full path
                   name.  Otherwise, just print the tree's name and
                   also display it's parent's name and version number */
        if ( p_opt )
        {
                printf("\n");
                        /* get the path name string */
                gpthname( treeptr, treetble, 0, 0, pathname );
                printf( "%s\n", pathname );

        }
        else
                printf ( "%s\tparent: %s %d.%d\n", treeptr->trename,
                        &treetble[treeptr->ptblink], treeptr->prntrel,
                        treeptr->prntdlta );

                /* If the user wants to know about the tree's releases
                   call 'prtrel' to print the info. */
        if ( r_opt )
                prtrel ( treeptr );

                /* If the user wants to know whether or not the tree
                   is reserved for change, look for an entry for the
                   tree in the change table.  If an entry is found,
                   then the tree is reserved */
        if ( c_opt )
        {
                if ( ( chgptr = chkopen( treeptr, chgtble, chgtbsz ) ) )
                        printf("Reserved for change by %s on %s",
                                chgptr->userid, ctime( &chgptr->time ) );
```

```
                else
                        printf( "NOT reserved for change\n" );
        }
                /* If the -d option was submitted, get the last revision
                   date. */
        if (d_opt)
        {
                relindx = treeptr->lastrel;
                if ( relindx >= 0 )
                {
                        comindx = rtbnodes[relindx].lasthis;
                        printf( "Last revision date:  %s",
                                ctime( &comtable[comindx].when  ) );
                }
        }
}
```

```
/* This is the first program executed in the 'put' operation. See 'put.doc'
   for more info on what the program does */

#include (stdio.h)
#include (time.h)
#include "defines.h"
#include "tablestr.h"
#include "contlstr.h"
#include "tabledef.h"
#include "idinfo.h"

#define error1 "'put' cannot find a new version of %s in the current directory\n", argv[1]

main( argc, argv )

int argc;
char *argv[];
{
        struct change *chgptr, *chkopen();
        char *getlogin();
        struct treeentry *treeptr, *getreptr(); /* tree pointers */
        int newrel;     /* new release indicator */
        char iobuff[buffsize];  /* i/o buffer */
        int commtsz;    /* number of characters in comment */
        int lnumbers;   /* line numbers indicator */
        int lines;      /* number of lines in old version */
        struct control contlrec; /*control record with new version numbers */
        int nonempty;   /* "tree not empty" indicator */
        int relnum, deltanum;   /* release and delta numbers */

        FILE *fp_mod, *fp_old, *fp_temp, *fp_hist, *fp_htmp;

                /* check that the required number of arguments have been
                   submitted */
        if ( argc < 3 )
        {       printf("module or tree name argument missing\n");
                exit(1);
        }

                /* check for the optional argument */
        if ( argc > 3 )
        {               /* check that a valid -r was submitted */
                if ( strcmp(argv[3], "-r" ) != 0 )
                {       printf("%s...invalid option\n", argv[3]);
                        exit (1);
                }
                else
                        newrel = 1;
        }
        else
                newrel = 0;

                /* check that the user actually has a new version
                   in the current directory */
        if ( fopen(argv[1], "r") == NULL )
        {       printf(error1);
```

```
            exit (1);
    }


                /* open the input and output files */
    if ( !openall(argv[1], &fp_mod, &fp_old, &fp_temp, &fp_hist, &fp_htmp) )
    {
                /* close all files */
            exit (1);
    }


            /* read the tables from the TCS file */
    readtble ( fp_mod, &tretblsz, &reltblsz, treetble, rtbnodes, &chgtbsz,
               chgtble);
            /* get the tree's pointer into the tree table */
            /* If the tree name is not found in the table, print error*/
    if ( !(treeptr = getreptr(argv[2],treetble)) )
    {       printf("%s...tree does not exist\n", argv[2]);
            closeall( fp_mod, fp_old, fp_temp, fp_hist, fp_htmp );
            exit (1);
    }


            /* Check that the tree is open for change. If it is also
               get it's pointer into the change table */
    if ( !( chgptr = chkopen(  treeptr, chgtble, chgtbsz )) )
    {       printf("%s is not open for change\n", argv[2]);
            closeall( fp_mod, fp_old, fp_temp, fp_hist, fp_htmp );
            exit (1);
    }


            /* Check the user ID */
    if ( strcmp( chgptr->userid, getlogin() ) != 0 )
    {       printf("Access denied; %s has %s open for change\n",
                    chgptr->userid, treeptr->trename );
            closeall( fp_mod, fp_old, fp_temp, fp_hist, fp_htmp );
            exit (1);
    }


            /* Get the comment from the user */
            /* 'getcommt' returns the comment size */
    commtsz = getcommt( iobuff, buffsize );


            /* set the new release field in the tree's table entry,
               if the new release argument was submitted */
    if ( newrel )
            treeptr->newrflag = newrel;


            /* Set the non-empty tree indicator. 'nonempty' will be
               0 if the tree is empty, greater than 0, otherwise */
    nonempty = treeptr->numrels;


            /* Update the tree and release tables with the new
               version number */
    if ( !( reltblsz = updtbls( treeptr, rtbnodes, reltblsz )) )
    {
            closeall( fp_mod, fp_old, fp_temp, fp_hist, fp_htmp );
            exit (1);
```

```
}

        /* Update the history file with the new version's
           comments */
if ( !updhist( fp_hist, chgptr->userid, commtsz, iobuff,
         treeptr, rtbnodes, fp_htmp ) )
{
        closeall( fp_mod, fp_old, fp_temp, fp_hist, fp_htmp );
        exit (1);
}

        /* The new release indicator is no longer needed.
           Turn it off */
treeptr->newrflag = 0;

        /* Set up the temporary file. The body of this file
           will have line numbered text */

        /* set up the control record */
contlrec.treenum = treeptr->trenum;     /* tree number */
contlrec.relnum = relnum = treeptr->numrels;    /* release number */
        /* delta number is the highest release's highest delta*/
contlrec.deltanum = deltanum = rtbnodes[treeptr->lastrel].hipdelta;

        /* save the version identification info */
        /* "putdelta" will use this for the 'success' message to
            the user */
sprintf( idbuff, "%s %d.%d\n", treeptr->trename, relnum, deltanum );
fputs( idbuff, fp_temp );
        /* Write the control record */
fwrite( &contlrec, sizeof(struct control), 1, fp_temp );
        /* Write the tree and release tables */
wrtrtble( &tretblsz, &reltblsz, treetble, rtbnodes, fp_temp );

        /* Write out the change table, deleting the tree's entry */
wrtchgtb( fp_temp, chgtble, chgtbsz, chgptr );

        /* Build the internal tree structure */
        /* If the tree is not empty, the tree's release and delta
           and tree numbers are passed to 'bldintst'. Otherwise,
           the parent's tree, release and delta numbers are passed */
if (nonempty)
        bldintst( treeptr, relnum, deltanum, rtbnodes, treetble,
          intrestr, trestrtb );
else
        bldintst( &treetble[treeptr->ptblink], treeptr->prntrel,
                  treeptr->prntdlta, rtbnodes, treetble,
                  intrestr, trestrtb );

        /* Get the old version and the line numbered text */
lnumbers = 1;   /* set the line numbered text indicator */
if ( !( lines = getversn(lnumbers, intrestr, trestrtb, fp_mod,
                         fp_old, fp_temp)) )
{
        printf( "%d lines retreived\n", lines );
        closeall( fp_mod, fp_old, fp_temp, fp_hist, fp_htmp );
```

```
                exit (1);
        }

        closeall( fp_mod, fp_old, fp_temp, fp_hist, fp_htmp );
        exit (0);
} /* end main */
```

```
/* Insert the deltas (output from 'diff') into the TCS module file */
#include (stdio.h)
#include "defines.h"
#include "tablestr.h"
#include "contlstr.h"
#include "tabledef.h"
#include "filedefs.h"
#include "idinfo.h"

#define msg1 "No difference was found between the old and new version....\n delta created\n"

main( argc, argv )
int argc;
char *argv[];
{
        FILE *fp_temp, *fp_diff, *fp_out;
        int fromline, toline; /* line number range limits for a delta */
        struct control contlrec, *contlptr;  /* control record and pointer */
        char *diffeof;  /* end of diff file indicator */
        char *append(), *delete(), *change();
        char dtype;     /* type of delta: append, delete, or change */
        char iobuff[buffsize];  /* i/o buffer */
        char diffbuff[difflnsz]; /* line from the diff file */


                /* Open all files */
        if ( !openf( argv[1], &fp_temp, &fp_diff, &fp_out) )
                exit (1);

                /* Read the identifying information for the new version */
        fgets ( idbuff, idbufsiz, fp_temp );
                /* Read the control record containing the tree, release,
                   and delta numbers of the new version.  The record is
                   in the temporary module file created by 'put' */
        fread( &contlrec, sizeof(struct control), 1, fp_temp);

                /* Read all tables */
        readtble( fp_temp, &tretblsz, &reltblsz, treetble, rtbnodes,
                  &chgtbsz, chgtble);

                /* Write out all tables */
        writabls( fp_out, &tretblsz, &reltblsz, treetble, rtbnodes,
                  &chgtbsz, chgtble);
                /* Process the body of the file, installing deltas */

                /* Read the first line of the diff file */
        if ( (diffeof = fgets( diffbuff, difflnsz, fp_diff )) == NULL)
        {               /* In this case, there are no differences in the
                           old and new version so reportthis and exit */
                printf( msg1 );
                exit (2);
        }

                /* Read the diff file, line by line, and install the
                   deltas in the new file */
        while( diffeof != NULL )
```

```
        {               /* Extract the line numbers affected and the
                           difference type: append, delete, or change.
                           The diff line being processed is sitting in
                           the 'diff' i/o buffer, at this point */
                difflt( diffbuff, &fromline, &toline, &dtype );

                switch (dtype){
                case 'a':       /* One or more lines are to be appended
                                   after 'fromline'. */
                        diffeof = append( fromline, &contlrec, fp_diff,
                                        fp_temp, fp_out, diffbuff, iobuff);
                        break;

                case 'd':       /* One or more lines are to be deleted */
                        diffeof = delete( fromline, toline, &contlrec,
                                  fp_diff, fp_temp, fp_out, diffbuff, iobuff);
                        break;

                case 'c':       /* One or more lines are changed */
                        diffeof = change( fromline, toline, &contlrec,
                                  fp_diff, fp_temp, fp_out, diffbuff, iobuff);
                        break;
                } /* end switch */
        } /* end while */

                /* All deltas have been installed; read the rest of the
                   temporary module file and write it out to the new file */
        rweof( fp_temp, fp_out, iobuff );

                /* Report success by sending the user the tree name and
                   version number of the new version */
        printf( "'put' created: %s", idbuff );

                /* Close all files and return the 'normal' completion
                   return code */
        closef( fp_diff, fp_temp, fp_out );
        exit (0);
} /* end putdelta */


openf( module, fp_temp, fp_diff, fp_out )
        /* Open all files used by 'putdelta' */

        /*input arguments */
char *module;   /* module name */
        /* output arguments */
FILE **fp_temp, **fp_diff, **fp_out;
{
        FILE *fopen();

                /* Open the temporary file containing the line numbered
                   version */
        if ( (*fp_temp = fopen( tmpfile2, "r" )) == NULL )
        {       printf( "'putdelta' cannot open %s\n", tmpfile2 );
                return (0);
```

```
            /* Open the file containing the 'diff' output */
        strcat( diffile, module );  /* get the diff file path name */
        if ( (*fp_diff = fopen( diffile, "r" )) == NULL )
        {       printf( "'putdelta' cannot open %s\n", diffile );
                return (0);
        }

                /* Open the temporary file which will contain the new
                   module with all deltas installed */
        strcat( tempfile, module );     /* Get the file's full path name */
        if ( (*fp_out = fopen( tempfile, "w" )) == NULL )
        {       printf( "'putdelta' cannot open %s\n", tempfile );
                return (0);
        }

        return (1);
}


closef( fp1, fp2, fp3 )
        /* close all files used by putdelta */
FILE *fp1, *fp2, *fp3;
{
        fclose(fp1);
        fclose( fp2 );
        fclose( fp3 );
}
```

```
/* Queue Manipulations */

#include "defines.h"
#include "contlstr.h"   /* control records and constants */

#define queuesz 101 /* queue size */
#define freesz  queuesz-1  /* number of free nodes in the queue */

        /* error messages */
#define error_1 ("control queue empty: error in 'getversn'\n" )
#define error_2 ("entry not found during queue search\n tree = %d \tversion \
%d.%d\n", contlptr->treenum, contlptr->relnum, contlptr->deltanum )
#define error_3 ("error in looking for a non-null entry: a 'yes' or 'no' flag \
was not found;\n" )


static
struct queuenod {
        int treenum;    /* treenumber from file */
        int relnum;     /* release number */
        int deltanum;   /* delta number */
        int flag;       /* flag */
        struct queuenod *nxtentry;  /* pointer to the next node in the queue */
}
queue[queuesz],*freelist[queuesz], *header; /* queue, freelist, queue header*/

static int top; /* index of the top of the free list */



initfree()  /* initialize the freelist */

/*  Set freelist[n] to the address of queue[n+1]  */
/*  0 ( = n ( queuesz */
/* also set the freelist top pointer to 0 */
{
        /* queueptr is used to get the address of each element in the queue */
        struct queuenod *queueptr;
        int i;

        /* initialize the pointer to the free list */
        /* initialize the freelist; place the address of queue[n] into */
        /* freelist[n] */
        for ( i = 0, queueptr = queue+1; i (= freesz-1; queueptr++, i++ )
                freelist[i] = queueptr;
        top = 0;                /* initialize the freelist top pointer */
} /* end initfree */


initque()    /*  initialize the header node for the queue */

{
                /* initially the queue is empty with the header node
                   pointing to itself */
        header = queue; /* the header node is the first element in 'queue'*/
        header->nxtentry = header;
```

```
                /* The flag field of the header node is set to 'no'.
                   This guarantees that 'takeline' is set to 'no'
                   at end of file and the last line is not taken twice*/
        header->flag = noflag;
} /* end initque  */


struct queuenod
*getnode()
        /* get the address of the next available node from the free list */
{
        /* first check that there is a node available */
        /* if there is no available node then the queue is full */

        if ( top == freesz )
        {
                printf ("queue full");
                return(0);      /* indicate the error */
        }
        /* return the next free node's address and adjust the top pointer */
        return (freelist[top++]);
} /* end getnode */


retnode(node)
        /* return a node to the free list */
        /* adjust the freelist top pointer */

struct queuenod *node;

{
        freelist[--top] = node;
} /* end retnode */


emptyque()
        /* test for queue empty condition */
        /* An empty queue is an error since this function is called by
           functions which expect an entry to be found */
{
        if (header->nxtentry == header )
        {
                printf( error_1 );
                return (true);
        }
        return (false);
}


enterque( contlptr, flag )
        /* place an entry at the head of the queue */
struct control *contlptr;       /* pointer to the control record */
int flag;       /* flag field value */

/* The function returns a 1, if the entry was successfully linked to the
```

```
   queue. Otherwise, it returns a 0. */

{
        struct queuenod *freenode;  /* pointer to the next free node */

                /* If a free node exists, get its address. Otherwise,
                   'freenode' is set to 0 and return */
        if ( (freenode = getnode() ) )
        {
                        /* Fill in the control information */
                freenode->treenum = contlptr->treenum;  /* tree number */
                freenode->relnum = contlptr->relnum;  /* release number */
                freenode->deltanum = contlptr->deltanum;  /*delta number */
                freenode->flag = flag;  /* flag field */
                        /* Link the node to the head of the queue */
                freenode->nxtentry = header->nxtentry;
                header->nxtentry = freenode;

                return (1);
        }
        else
                return (0);
}


leavequeue( contlptr )
        /* Remove the entry that matches the tree, release, and delta on
           the control record */
struct control *contlptr;   /* pointer to the control record */


{
        struct queuenod *queueptr;  /* used to traverse the queue */

                /* Search the queue sequentially, until the entry is found.
                   the search should always be succesful.*/
        queueptr = header; /* queueptr always points to the last node
                              examined */

        while ( (queueptr->nxtentry)->treenum != contlptr->treenum  ||
             (queueptr->nxtentry)->relnum != contlptr->relnum  ||
             (queueptr->nxtentry)->deltanum != contlptr->deltanum )
        {
                queueptr = queueptr->nxtentry;  /* point to the next entry */
                        /* If the search cycles back to the header node
                           then the entry wasn't found.  This is a fatal
                           error...exit immediately */
                if ( queueptr == header )
                {
                        printf(error_2);
                        exit (1);
                }
        }

                /* The entry was found; remove it. */
        retnode( queueptr->nxtentry ); /* return the node
```

```
                                        to the free list */

                /* delink the node from the queue */
        queuenptr->nxtentry = (queuenptr->nxtentry)->nxtentry;
}


headflag()
        /* get the flag field of the entry at the head of the queue */
{
                return ( (header->nxtentry)->flag );
}


fndnonul()
        /* Find the first non-null entry in the queue. The search should
           always be successful. */

/* the function returns the flag field of the first non-null entry. If the
   search winds back at the header node, the header's flag which is set to
   'no' is returned ( this happens at end of file ) */

{
        struct queuenod *ptr;  /* 'ptr' traverses the queue */

        for( ptr = header->nxtentry; ptr->flag == nullflag;
             ptr = ptr->nxtentry )
                ;

        return (ptr->flag);
}
```

```
/* Read the tree and release tables from an external file */
/* This function does not open the file but accepts a file pointer
   from its argument list */

#include (stdio.h)
#include "defines.h"
#include "tablestr.h"

readtble( fp, tsize, rsize, treetble, reltable, csize, chgtble )

        /* input argument */
FILE *fp;       /* file pointer */
        /* output arguments */
int *tsize;     /* size of tree table */
int *rsize;     /* size of release table */
struct treeentry *treetble;     /* tree table */
struct relnode *reltable;       /* release table */
int *csize;     /* change table size */
struct change *chgtble; /* change table */

{



        fread( tsize, sizeof(*tsize), 1, fp);
        fread(rsize, sizeof(*rsize), 1, fp);
        fread(treetble, sizeof(struct treentry), *tsize, fp);
        fread( reltable, sizeof(struct relnode), *rsize, fp);
        fread( csize, sizeof(*csize), 1, fp );
        fread( chgtble, sizeof(struct change), *csize, fp );

}
```

```
/* Read the tree and release tables from an external file */

#include (stdio.h)
#include "defines.h"
#include "tablestr.h"

readtbls( filename, tsize, rsize, treetble, reltable )

char *filename;
int *tsize;
int *rsize;
struct treeentry *treetble;
struct relnode *reltable;

{
        FILE *fopen(), *fp;


        if ( ( fp = fopen(filename, "r") ) == NULL )
        {       printf( "cannot open %s ", filename );
                exit (1);
        }

        fread( tsize, sizeof(*tsize), 1, fp);
        fread(rsize, sizeof(*rsize), 1, fp);
        fread(treetble, sizeof(struct treentry), *tsize, fp);
        fread( reltable, sizeof(struct relnode), *rsize, fp);

        fclose (fp);
  }
```

```
/* Read the 'putdelta' input file and write it out to the new temporary
   module until the end of the input file is hit */

#include (stdio.h)
#include "defines.h"
#include "contlstr.h"

rwbody( fp_temp, fp_out, iobuff )

FILE *fp_temp, *fp_out;
char *iobuff;
{
        int lineno;     /* line number of a text line in the input file */
        struct control contlrec;        /* control record */
        char type;      /* control character type */

        while ( (type = getc(fp_temp)) != EOF )
        {
                        /* write out the control character */
                putc( type, fp_out );

                switch (type){
                case inscontl:
                case delcontl:
                case endcontl:
                        /* The control character is an insert, delete,
                           or end control; read the control record */

                        fread(&contlrec, sizeof(struct control), 1, fp_temp);
                                /* Write out the control record */
                        fwrite(&contlrec, sizeof(struct control), 1, fp_out);
                        break;

                case txtcontl:
                                /* A text line follows the control record */
                                /* read and write the text line */
                        fgets(iobuff, buffsize, fp_temp );
                        fputs( iobuff, fp_out);
                        break;
                } /* end switch */
        } /* end while */
}
```

```
/* Read the 'putdelta' input file and write it out to the new temporary
   module until the end of the input file is hit */

#include (stdio.h)
#include "defines.h"
#include "contlstr.h"

rweof( fp_temp, fp_out, iobuff )

FILE *fp_temp, *fp_out;
char *iobuff;
{
        int lineno;     /* line number of a text line in the input file */
        struct control contlrec;        /* control record */
        char type;      /* control character type */

        while ( (type = getc(fp_temp)) != EOF )
        {
                        /* write out the control character */
                putc( type, fp_out );

                switch (type){
                case inscontl:
                case delcontl:
                case endcontl:
                        /* The control character is an insert, delete,
                           or end control; read the control record */

                        fread(&contlrec, sizeof(struct control), 1, fp_temp);
                                /* Write out the control record */
                        fwrite(&contlrec, sizeof(struct control), 1, fp_out);
                        break;

                case txtcontl:
                                /* A text line follows the control record */
                                /* Read and ignore the line number */
                        fread( &lineno, sizeof(lineno), 1, fp_temp );
                                /* read and write the text line */
                        fgets(iobuff, buffsize, fp_temp );
                        fputs( iobuff, fp_out);
                        break;
                } /* end switch */
        } /* end while */
}
```

```
/* Read records from the temporary line numbered file and write out
   records to the temporary new module file with text line numbers
   deleted. */

#include (stdio.h)
#include "defines.h"
#include "contlstr.h"

rwfile( up_to, writincl, fp_temp, fp_out, iobuff )

        /* input arguments */
int up_to;  /* read and write up to this line number */
int writincl; /* indicates whether or not line 'up_to'should be written out*/
FILE *fp_temp;  /* Line numbered input file */
        /* output arguments */
FILE *fp_out;   /* temporary new module output file */
char *iobuff;   /* i/o buffer for text lines */
{
        int lineno;     /* line number read from the input file */
        int lastline;   /* indicates that 'up_to' line has been processed*/
        struct control contlrec;        /* control record buffer */
        char contltyp;  /* control record type buffer */

        lastline = 0;
        while ( lastline != up_to )
        {
                        /* read the control character */
                contltyp = getc(fp_temp);
                        /* Process the line following the control char */
                switch (contltyp){
                case inscontl:
                case delcontl:
                case endcontl:
                                /* A control record follows. Read it in and
                                   write it out */
                        fread( &contlrec, sizeof(struct control), 1, fp_temp);
                        wrtcontl( contltyp, &contlrec, fp_out );
                        break;

                case txtcontl:
                                /* A text line follows the control character*/
                                /* First read the line number */
                        fread( &lineno, sizeof(lineno), 1, fp_temp );
                                /* read the line */
                        fgets( iobuff, buffsize, fp_temp );
                                /* If we haven't reached the 'up_to' line yet,
                                   write the line out.  Or if we have and the
                                   'writincl' indicator is set, write the line
                                   out.  Otherwise, just leave the line in the
                                   i/o buffer and let the caller decide what
                                   to do with it */
                        if ( lineno ( up_to  ||
                                (lineno == up_to  &&  writincl) )
                        {
                                putc( txtcontl, fp_out );
                                fputs( iobuff, fp_out );
```

```
                        }
                        lastline = lineno;
                        break;
                } /* end switch */
        } /* end while */
} /* end rwfile */
```

```
/* Read the body of the history file and write it out to the temporary
   new history file */

#include (stdio.h)
#include "defines.h"
#include "histstr.h"

rwhistf( comtable, numentry, fp_hist, fp_htmp )

        /* input arguments */
struct cominfo comtable[];      /* comment table */
int numentry;   /* number of comments to be written out */
FILE *fp_hist;  /* history file input pointer */
        /* output arguments */
FILE *fp_htmp; /* temporary new history output file */
{
        struct cominfo *ptr;    /* used to traverse the comment table */
        struct cominfo *limit;  /* pointer to the last comment table entry */
        char iobuff[buffsize];  /* i/o buffer */

        for ( ptr = comtable, limit = comtable + numentry; ptr ( limit; ptr++ )
        {
                        /* read a comment */
                fread( iobuff, sizeof(*iobuff), ptr->blksize, fp_hist );
                        /* Write out the comment */
                fwrite( iobuff, sizeof(*iobuff), ptr->blksize, fp_htmp );
        }
 }
```

```
        /* set the control flag */

#include "defines.h"
#include "tablestr.h"
#include "contlstr.h"

setflag( contlptr, intrestr, trestrtb, type )

struct control *contlptr;       /* pointer to the control record */
int intrestr[]; /* internal tree structure */
struct refs trestrtb[]; /* cross reference table for the internal tree struct.*/
{
        int apply;      /* indicates whether or not delta should be applied*/

                /* search the tree structure cross reference table for the
                   tree number. If the tree number is found, check the release
                   release and delta numbers.If the tree is not found, the
                   delta should not be applied */

        for ( ; trestrtb->trenum ) 0 && trestrtb->trenum != contlptr->treenum;
                trestrtb++ );

        if ( contlptr->treenum == trestrtb->trenum )  /* was the tree found*/
                apply = chkrd( contlptr, intrestr, trestrtb );
        else
                apply = false;

        switch (apply){
        case true:
                switch (type){
                case inscontl:  /* is this an insert? */
                        return (yesflag);  /*set the flag to apply the delta*/
                case delcontl:  /* is this a deletion? */
                        return (noflag);  /* set flag to "do not" apply delta*/
                }
         case false:
                switch (type){
                case inscontl:
                        return (noflag); /* set flag to "do not apply" */
                case delcontl:
                        return (nullflag); /* set the flag to "null" */
                }
         }
  }

        /* check to see if the release and delta should be included in
           the version retreived */

  chkrd(contlptr, intrestr, tresptr)

  struct control *contlptr;       /* pointer to control record */
  int intrestr[]; /* internal tree structure */
  struct refs *tresptr;   /* pointer to cross reference table entry for
                             the tree in question */


  {
```

```
        int i, apply;

        if ( contlptr->relnum (= tresptr->numrels )
                        /* the release is to be included. Calculate the
                        index into the internal tree structure for this
                        tree and check to see if the delta should be
                        included. */
        {
                i = tresptr->trestndx + contlptr->relnum - 1;
                if ( contlptr->deltanum (= intrestr[i] )
                        return (true);  /* the delta should be applied */
        }
        return (false);  /* The delta should not be applied*/
}


        /* set the "take line" indicator */
setkline( flag )

        /* input argument */
int flag;

/* The function returns a value indicating whether the next line of next
   should or should not be taken. */

{
        switch (flag){
        case yesflag:
        case noflag:
                        /* A 'yes' value in 'flag' indicates that the
                           line is to be taken. A 'no' in 'flag' means
                           don't take the line. */

                return (flag);
        case nullflag:
                        /* Get the flag field of the first non-null entry
                        entry in the queue. If the queue is empty or a
                        non-null entry could not be found, then something's
                        wrong and 'fndnonul' will be 0. */

                return ( fndnonul() );

        }
}
```

```
/* Show information regarding a TCS module file */
/* See "show.doc" for USAGE documentation */

#include (stdio.h)
#include "defines.h"
#include "tablestr.h"
#include "filedefs.h"
#include "histstr.h"
#include "tabledef.h"
#include "histdefs.h"

main( argc, argv )

int argc;
char *argv[];
{

        FILE *fp, *fp_hist, *fopen();
        int rtcode;  /* return code */

                /* open the TCS module file */
        strcat ( modname, argv[2] );
        if ( (fp = fopen( modname, "r" )) == NULL )
        {
                printf( "'SHOW' cannot open %s\n", modname );
                exit (1);
        }

                /* Open the history file */
        strcat( histfile, argv[2] );
        if ( (fp_hist = fopen( histfile, "r")) == NULL )
        {
                printf("'SHOW' cannot open %s\n", histfile);
                exit (1);
        }

                /* Read the tree, release, and change tables
                   in the TCS file */

      readtble( fp, &tretblsz, &reltblsz, treetble, rtbnodes,
                  &chgtbsz, chgtble );

                /* Read the comment table from the history file */
                /* first read the table size */
      fread( &hstsizes, sizeof(struct hsizes), 1, fp_hist );
      fread( comtable, sizeof(struct cominfo),
                hstsizes.comtbsz, fp_hist );

                /* Get the info that the user requested */
     if ( strcmp( argv[1], "tree" ) == 0 )
     {
                        /* The user wants tree information */
                rtcode = showtree( argc, argv, fp, fp_hist );
                exit (rtcode);
     }
```

```
        if ( strcmp( argv[1], "child" ) == 0 )
        {
                        /* The user wants to know what trees are children
                           of the given node */
                rtcode = showchil( argc, argv, fp, fp_hist );
                exit ( rtcode );
        }
                /* If we get up to here, then there is no info something's
                   wrong. */
        printf( "'show' has no information on %s\n", argv[2] );
        printf( "USAGE: show  (show function) (module name)  [options]\n" );
        exit (1);
}
```

```
/* show children trees of the given node */
/* See "show.doc" for USAGE documentation */

#include (time.h)
#include (stdio.h)
#include "defines.h"
#include "tablestr.h"
#include "histstr.h"
#include "showextn.h"

#define num_carg 5


#define chg_msg "%s\treserved for change by %s on %s\n", tbleptr->trename, \
chgptr->userid, ctime( &chgptr->time )

#define chg_msg2 "%s\tNOT reserved for change\n", tbleptr->trename

#define childcnt "%d child trees\n", cnt

#define parn_msg "%s %d.%d child trees:\n", treeptr->trename, relnum, deltanum

char childopt[] = "t:v:c";      /* valid options for "show child"*/

showchil( argc, argv, fp, fp_hist )

       /* input arguments */
int argc;
char *argv[];
FILE *fp, *fp_hist;

/* The function returns a 0 if no errors were encountered in getting
   the requested information.  Errors are mostly user input errors in
   the command line */
{
       int showind[num_carg];  /* optional arguments indicators */
       int i;
       int c, cnt;
       int relnum;     /* release number */
       int deltanum;   /* delta number */
       int treeindx;   /* index of tree's entry in the tree table */
       extern int nxtarg;      /* set by tcsopt() */
       extern char *optnarg;   /* set by tcsopt() */
       struct change *chgptr, *chkopen();  /* change tble pointers */
       struct relnode *rdptr;  /* need this for call to fndtrdnm() */
       struct treentry *treeptr;       /* pointer to tree in tree table */
       struct treentry *emptyptr;  /* pointer to tree if the tree is empty */
       struct treentry *tbleptr;  /* used to traverse the tree table */
       char *treename; /* pointer to tree name on command line */
       char *vername;  /* pointer to version number on command line */
       char *ctime();

              /* Initialize the optional argument indicators */
       for ( i = 0; i ( num_carg; i++ )
              showind[i] = false;
              /* get the optional arguments and set their indicators */
```

```
                /* Options start at argv[3] */
    while ( ( c = tcsopt( argc - 3, argv+3, childopt ) ) != EOF )
    {

            switch (c){
                    case 't':
                            showind[treearg] = true;
                            treename = optnarg;
                            break;

                    case 'v':
                            showind[versarg] = true;
                            vername = optnarg;
                            break;

                    case 'c':
                            showind[changarg] = true;
                            break;

                    case '?':
                            return (1);

            } /* end switch */
    } /* end while */

            /* get the release and delta numbers in integer form
               and also the tree pointers. */
    if ( !fndtrdnm( treename, vername, rtbnodes, treetble,
             showind, &relnum, &deltanum, &treeptr,
            &emptyptr, &rdptr ) )
            return (1);

            /* Search the tree table for all trees that are
               children of this node.  If the parent tree is
               empty then just print a message and return. */

    if ( emptyptr )
    {
            printf("'show child':  %s...no children; tree is empty\n");
            return (1);
    }

            /* Get the node's index into the tree table */
    treeindx = treeptr - treetble;
            /* Print out the parent tree, release, and delta */
    printf( parn_msg );
            /* Keep a count of the children, in 'cnt' and print it
               at the end */
    for ( cnt = 0, tbleptr = treetble + tretblsz - 1;
            tbleptr ) treetble; tbleptr-- )
    {
           if ( tbleptr->ptblink == treeindx  &&
                   tbleptr->prntrel == relnum &&
                   tbleptr->prntdlta == deltanum )
           {
                           /* Print the child's name */
```

```
                cnt++;  /* count the child */

                if ( showind[changarg] )
                {
                        if( (chgptr = chkopen( tbleptr, chgtble,
                                                chgtbsz) ) )
                                printf( chg_msg );
                        else
                                printf( chg_msg2 );
                }
                else
                        printf( "%s\n", tbleptr->trename );
        }
    }
    printf( childcnt );
    return (0);
}
```

```
/* Show information about one or more trees */
/* See "show.doc" for USAGE documentation */

#include (stdio.h)
#include "defines.h"
#include "tablestr.h"
#include "histstr.h"
#include "showextn.h"   /* external table definitions */
#include "shotrdef.h"

showtree( argc, argv, fp, fp_hist )

        /* input arguments */
int argc;
char *argv[];
FILE *fp, *fp_hist;

/* The function returns a 0 if no errors were encountered in getting
   the requested information.  Errors are mostly user input errors in
   the command line */
{
        int showind [sho_argn]; /* optional arguments indicators */
        struct treentry *treeptr;       /* pointer to tree in tree table */
        struct treentry *getreptr();
        char *treename; /* pointer to tree name argument if -t submitted*/

                /* Get the optional arguments, set the options indicators,
                   and reteive the tree name argument pointer */
        if (streeopt( argc, argv, showind, &treename ))
                return (1);
        if ( t_opt )
        {
                /* Information about a specific tree is requested.
                   get the tree pointer and print the requested info */

                if ( !(treeptr = getreptr(treename, treetble)) )
                {
                        printf("%s...tree does not exist\n", treename);
                        return (1);
                }
                else
                {
                        prtrinfo( treeptr, showind, fp, fp_hist );
                        return (0);
                }
        }

        if ( a_opt )
        {
                        /* Go through the tree table and print the
                           requested info for each tree */

                        /* The table is scanned backwards, so that the
                           info for the "latest created" trees is
                           printed first. */
                for( treeptr = treetble + tretblsz -2; treeptr )= treetble;
```

```
                    treeptr-- )
                    prtrinfo( treeptr, showind, fp, fp_hist );
            return (0);
        }
        return (0);
}
```

```
/* Get the options for the 'SHOW TREE' function.  Set the option
   indicators and do some error checks on the options */

#include (stdio.h)
#include "defines.h"
#include "tablestr.h"
#include "shotrdef.h"

#define opt_err1 "option error: incompatible options \
-a and -t submitted\n"


#define opt_err3 "option error: required option -a or -t missing\n"


char treeopts[] = "t:aprcde";
streeopt( argc, argv, showind, treename )
        /* input arguments */
int argc;
char *argv[];
        /* output arguments */
int showind[];  /* optional argument indicators */
char **treename;        /* pointer to the tree name argument */
/* the function returns a 1 if some error in the command line is
   encountered.  Otherwise, it returns a 0. */

{
        int c;
        int i;
        extern int nxtarg;      /* set by tcsopt() */
        extern char *optnarg;   /* set by tcsopt() */

                /* Initialize the optional argument indicators */
        for ( i = 0; i ( sho_argn; i++ )
                showind[i] = false;

                /* Get the options */
                /* Options for this function start at argv[3] */
        while ( ( c = tcsopt( argc - 3, argv+3, treeopts ) ) != EOF )
        {
                switch (c){
                        case 'a':
                                if ( t_opt )
                                {
                                        printf( opt_err1 );
                                        return (1);
                                }
                                else
                                        a_opt = true;
                                break;

                        case 't':
                                if ( a_opt )
                                {
                                        printf( opt_err1 );
                                        return (1);
```

```
                        }
                        else
                        {
                                t_opt = true;
                                  /* assume that the tree name
                                     follows the -t option.  Get
                                     its pointer */
                                *treename = optnarg;
                        }
                        break;

                case 'e':
                        p_opt = true;
                        c_opt = true;
                        r_opt = true;
                        d_opt = true;
                        break;

                case 'p':
                        p_opt = true;
                        break;

                case 'r':
                        r_opt = true;
                        break;

                case 'c':
                        c_opt = true;
                        break;

                case 'd':
                        d_opt = true;
                        break;

                case '?':
                        return (1);
            }  /* end switch */

        }  /* end while */
                /* make sure that the -a or the -t was submitted */
        if ( !a_opt && !t_opt )
        {
                printf( opt_err3 );
                return (1);
        }

        return (0);
}
```

```
/* Get the next option letter in a TCS command line */

#include (stdio.h)

#define charmask 255 /* mask to make characters positive */

#define opt_err1 "option error...Invalid option '%c' in %s\n", \
argptr[nxtchar], argv[nxtarg]

#define opt_err2 "option error... option '%c' requiring an argument \
submitted in %s\n", *optstr, argv[nxtarg]

char *optnarg = 0;      /* pointer to an option argument */
        /* index of next character to be examined within the next
           argument to be examined */
int nxtchar = 0;
        /* 'argv' index of the next argument to be examined */
int nxtarg = 0;

tcsopt( argc, argv, optstr )

int argc;       /* total number of arguments examined */
char *argv[];   /* pointers to arguments */
char *optstr;   /* valid options string */

{
        char *argptr;   /* pointer to the option string being examined */

                /* If 'argc' is 0 to begin with, there are no options
                   to process.  So just return EOF */
        if ( !argc )
                return (EOF);

                /* while there is another option letter, retieve it */
        while ( nxtarg < argc )
        {
                        /* Get the pointer to the next argument
                           string to be examined */
                argptr = argv[nxtarg];

                        /* If 'nxtchar' = 0 then we're processing the
                           a new argument string. */
                if ( nxtchar == 0 )
                {
                        /* Check to see if the end of the options is hit */

                                /* If the first character is not '-',
                                   then we're done */
                        if ( argptr[nxtchar] != '-' )
                                return (EOF);

                        nxtchar++;      /* point to the second character */

                                /* If a '-' by itself was submitted,
                                   then we're done */
                        if ( argptr[nxtchar] == '\0' )
```

```
                        return ( EOF );

                        /* If a '--' was submitted, this marks
                           the end of the options */
                    if ( argptr[nxtchar] == '-' )
                    {
                                /* Ignore the '--' but point to the
                                   next argument */
                        nxtarg++;
                        nxtchar = 0;
                        return (EOF);
                    }
                        /* An end of options condition has not
                           been hit yet; so process the option
                           letter. */
                    return ( optbegn( argv, argptr, optstr ));
                }

                else
                {
                        /* The last pass through this function left off
                           in the middle of a 'no argument' letter string.
                           So, continue to look for 'no argument letters.
                           If the end, of the previous string is hit, then
                           go onto the next option string */

                        if ( argptr[nxtchar] == '\0' )
                        {
                                nxtarg++;
                                nxtchar = 0;
                        }
                        else
                                /* look for a valid option letter */
                                /* Since we're in the middle of a
                                   'no argument' letter string, a ':'
                                   following the option found, is an
                                   error */

                                return ( noargopt( argv, argptr, optstr ));
                }
        }
                /* If we fall out of the while loop, then we've
                   hit the end of the options */
        return ( EOF );
}


optbegn( argv, argptr, optstr )

char *argv[];
char *argptr;   /* pointer to the option to be examined */
char *optstr;   /* pointer to the valid option string */
{
        for ( ; *optstr != '\0'; optstr++ )
        {
                        /* See if we have a valid argument */
```

```
                if ( argptr[nxtchar] == *optstr )
                {
                        /* If a ':' was found in the input
                           string, then signal the error */
                    if ( *optstr == ':' )
                        break;

                        /* The input option letter is valid*/
                        /* Point to the next character in
                           the input string */
                    nxtchar++;
                        /* See if the option letter should
                           have an argument following */
                    if( *(optstr + 1) == ':' )
                    {
                        /* An option argument follows; get its
                           pointer.  The option letter and its
                           argument can be separated by white
                           space or not. */

                        if ( argptr[nxtchar] == '\0' )
                                /* White space follows */
                            optnarg = argv[++nxtarg];
                        else
                                /* No white space follows */
                            optnarg = argptr + 2;
                            /* Point to the next argument
                               to be processed */
                        nxtarg++;
                            /* reset the character index for the
                               next argument to be processed */
                        nxtchar = 0;
                    }
                    return ( *optstr & charmask );
                }
        }
                /* If we fall thru the 'for' loop, then a valid option was
                   not found.  point to the next character in the option
                   input string and return '?' */
        printf( opt_err1 );
        nxtchar++;
        return ( '?' & charmask );
}


noargoot( argv, argptr, optstr )

char *argv[];
char *argptr;   /* pointer to the option to be examined */
char *optstr;   /* pointer to the valid option string */
{
        for ( ; *optstr != '\0'; optstr++ )
        {
                        /* See if we have a valid argument */
                if ( argptr[nxtchar] == *optstr )
                {
```

```
                        /* If a ':' was found in the input
                           string, then signal the error */
                    if ( *optstr == ':' )
                        break;

                        /* The input option letter is valid*/
                        /* Point to the next character in
                           the input string */
                    nxtchar++;
                        /* See if the option letter should
                           have an argument following */
                    if( *(optstr + 1) == ':' )
                    {
                        /* The input string should not have
                           any 'options requiring an argument */

                        printf( opt_err2 );
                        return ('?' & charmask );
                    }

                        /* the option is OK; reutrn it */
                    return ( *optstr & charmask );
                }
            }
                /* If we fall out of the 'for' loop , an invalid option
                   was found in the input string */
            printf( opt_err1 );
            nxtchar++;
            return( '?' & charmask );
        }
```

```
/* Update the comment table and the release table with the index of the
   release's last comment entry */


#include (time.h)
#include "defines.h"
#include "tablestr.h"
#include "histstr.h"

#define error1 "'updcomtb' cannot update the comment table...comment table full...\nIncrease 'numvers' in

updcomtb( userid, blksize, treeptr, reltable, hstsizes, comtable )

       /* input arguments */
char *userid;   /* user ID */
int blksize;    /* comment block size */
struct treentry *treeptr;       /* tree pointer */
       /* output arguments */
struct relnode reltable[];      /* release table */
struct hsizes *hstsizes;        /* comment table and history sizes */
struct cominfo comtable[];      /* comment table */

/* The function returns a 1 if the comment table was successfully
   updated.  It returns a 0, otherwise */

{
       long time();
       int ctbsize;    /* current comment table size */
       struct cominfo *newnode; /* pointer to new comment table entry */
       struct relnode *rdptr;  /* pointer to tree's last release node */
       char iobuff[buffsize];  /* i/o buffer */

              /* If there's no room left in the comment table, return a
                 0. Otherwise, update the table */
       if ( (ctbsize = hstsizes->comtbsz) == numvers )
       {
              printf( error1 );
              return (0);
       }

       newnode = comtable + ctbsize;   /* get the new node's pointer */

              /* Get the pointer to the tree's last release */
       rdptr = reltable + (treeptr->lastrel);

              /* Link the new node to the end of the comment table */

       if ( rdptr->lasthis < 0 ) /* Are there any comments for this release? */
                     /* The new node is the first comment table entry
                        for this release. Link it at the head of the list*/
              rdptr->firsthis = ctbsize;
       else
                     /* Link the node to the end of the release's
                        comment entries */
              comtable[rdptr->lasthis].nxtentry = ctbsize;
```

```
        /* Mark the end of the comments for this release */
    newnode->nxtentry = 0;
        /* Store the index of the last comment entry for this
           release */
    rdptr->lasthis = ctbsize;

        /* Put the comment information into the new node */
    time(&newnode->when);   /* Fill in the time and date */
    newnode->blksize = blksize; /* Fill in the comment block size */
        /* The first character of the comment is on character
           beyond the present total history size. */
    newnode->offset = hstsizes->histsz;
    strcpy( newnode->userid, userid ); /* fill in the user id */

    hstsizes->comtbsz = ++ctbsize;  /* store the new comment table size*/
    hstsizes->histsz += blksize;    /* store the new history size */

    return (1);
}
```

```
/* Update the history file with the new comment */
/* An entry for the comment is placed at the end of the comment table and
   the comment itself is placed at the end of the file */

#include (stdio.h)
#include (time.h)
#include "defines.h"
#include "tablestr.h"
#include "histstr.h"
#include "histdefs.h"

updhist( fp_hist, userid, blksize, comment, treeptr, reltable, fp_htmp )

        /* input arguments */
FILE *fp_hist;  /* history file */
char *userid;   /* user ID */
int blksize;    /* comment block size in characters */
char *comment;  /* the comment itself */
struct treentry *treeptr;       /* tree's tree table entry pointer */
struct relnode reltable[];      /* release tables */
        /* output arguments */
FILE *fp_htmp;  /* temporary new history file */
{

        int numc;       /* number of comments in the current file */
                /* Read the comment table and history block size */
        fread( &hstsizes, sizeof(struct hsizes), 1, fp_hist );
                /* Read the comment table */
                /* save the number of entries in the current table */
        numc = fread( comtable, sizeof(struct cominfo),
                hstsizes.comtbsz, fp_hist );

                /* Update the comment table */
        if ( !updcomtb( userid, blksize,treeptr, reltable, &hstsizes, comtable))
                return (0);

                /* Write out the new comment table and history sizes */
        fwrite( &hstsizes, sizeof(struct hsizes), 1, fp_htmp );
                /* Write out the new comment table */
        fwrite( comtable, sizeof(struct cominfo), hstsizes.comtbsz, fp_htmp );

                /* Read the body of the history file and write it out */
        rwhistf( comtable, numc, fp_hist, fp_htmp );

                /* Write out the new comment */
        fwrite( comment, sizeof(*comment), blksize, fp_htmp );

        return (1);
}
```

```
/* Update the tree and release tables with the new version numbers */

#include "defines.h"
#include "tablestr.h"

#define error1 ("'put' cannot update the release table.\nIncrease 'maxrnode' in 'defines.h'.\n")

updtbls( treeptr, rtbnodes, rsize )

struct treentry *treeptr;       /* pointer to the tree's table entry */
struct relnode rtbnodes[];      /* release tables */
int rsize;      /* release tables array size */

/* The function returns the new release tables' size if the tables were
   successfully updated.  Otherwise it returns 0. */
{

                /* If the tree is flagged to have a new release,
                   add 1 to the number of releases for the tree,
                   add a new release node to the tree's release table,
                   and set the delta in the release node to 1. */

                /* If the new release flag is not set, add 1 to the
                   highest delta in the highest release node for the tree */

        if ( treeptr->newrflag )
        {
                if ( newrelse( treeptr, rtbnodes, rsize) )
                                return (0);
                else
                        rsize++; /* Count the new node. */
        }
        else
                        /* increment the highest delta field
                           in the tree's last release node */
                (rtbnodes[treeptr->lastrel].hipdelta)++;

        return (rsize);
}   /* end updtbls */

newrelse( treeptr, rtbnodes, rsize)
        /* input and output arguments */
struct treentry *treeptr;       /* tree table pointer */
struct relnode rtbnodes[];      /* release tables */
int rsize;      /* release table size */
{

                        /* add a new release node */

        if ( rsize == maxrnode ) /* is there room in the release
                                    table? */
        {       printf(error1);
                return (1);
        }

                /* The next available node is at rtbnodes[rsize] */
```

```
        /* fill in the new release node info */
    rtbnodes[rsize].hipdelta = 1;   /* set the highest delta */
        /* indicate an empty history table for the release */
    rtbnodes[rsize].firsthis = -1;
    rtbnodes[rsize].lasthis = -1;
    rtbnodes[rsize].nxtrnode = 0; /* mark the end of the list */

        /* Link the node to the release table */
    if (treeptr->numrels) /* is this a non-empty tree? */
        rtbnodes[treeptr->lastrel].nxtrnode = rsize;
    else
        treeptr->firstrel = rsize;

        /* set the pointer to the tree's last release node */
    treeptr->lastrel = rsize;

    treeptr->numrels++; /* increment the number of releases */
    return (0);
}
```

```
/* Update the tree table. Enter a new tree in the table */

#include "defines.h"
#include "tablestr.h"

#define error_1 "'updtrtbl'....Tree table full....\nIncrease 'maxtrees' in 'defines.h'"
updtrtbl( newtree, parmptr, prelse, pdelta, tsize, treetble )

        /* input arguments */
char *newtree;  /* new tree name */
struct treentry *parmptr;       /* parent tree point into the tree table */
int prelse;     /* parent's release number */
int pdelta;     /* parent's delta number */
int tsize; /* tree table size */
        /* output arguments */
struct treentry treetble[];     /* tree table */

/* The function returns the new tree table size if the entry was successfully
   entered.  Otherwise, it returns a 0. */

{

        struct treentry *treeptr;  /* new treeptr */

                /* Make sure that there's room in the tree table */
        if ( tsize == maxtrees )
        {       printf( error_1 );
                return (0);
        }

                /* point to the end of the tree table */
        treeptr = treetble + tsize - 1;

                /* Fill in the new tree's entry */
        strcpy( treeptr->trename, newtree); /* fill in the tree name */
        treeptr->trenum = tsize;  /* fill in the tree number */
                        /* figure out the parent's index and fill it in */
        treeptr->ptblink = parmptr - treetble;
        treeptr->prntrel = prelse; /* fill in the parent release */
        treeptr->prntdlta = pdelta; /* fill in the parent delta */
                        /* indicate and empty tree by next group of settings*/
        treeptr->numrels = 0; /* number of releases */
        treeptr->firstrel = -1; /* indicate an empty release table */
        treeptr->lastrel = -1;
                        /* Set the new release flag to force the first
                           'put' operation to create version 1.1 */
        treeptr->newrflag = 1;

                        /* Mark the end of the tree table */
        tsize++;  /* increment the tree table size */
        strcpy( treetble[tsize].trename, "" );

        return (tsize);
}
```

```
#include "ends.c"

copy(new_str, string)
        /* copy a character string (whose pointer is "string") into */
        /* another string (whose pointer is new_string) */


char *new_str, *string;
{

        while (*new_str++ = *string++);
}

strlen(string)          /* get the length of a character string */

char *string;

{
int  size;      /* character counter */

        for ( size = 0; *string++ ; size++ );
        return (size);
end
```

```
/* verify that the version given by the release and delta (relnum,deltanm) */
/* exist in the tree, given by treenum. VERIFYRD returns the pointer to the  */
/* release if the version exists, 0 otherwise                                 */

#include "defines.h"
#include "tablestr.h"

struct relnode
*verifyrd(treeptr,treetble,rtbnodes,relnum,deltanm)

struct treentry *treeptr;           /* tree pointer */
struct treentry treetble[]; /* tree table */
struct relnode rtbnodes[];  /* release table nodes */
int relnum;                 /* release number submitted */
int deltanm;                /* delta numbers submitted */

{
        int relindx;   /* traverses the list of release nodes */

        /* if the release number is equal to 0 or greater than the tree's */
        /* highest release or the delta number =0, then immediately return 0*/

        if ( relnum (= 0 || relnum ) treeptr-)numrels || deltanm (= 0)
                return (0);

        /* the release number is valid; check the delta */
        /* First, get the index into the release table for the release */

        relindx = getrindx( treeptr, rtbnodes, relnum );

        /* if the delta number submitted is greater then both the */
        /* highest propagating delta, then the delta does not exist */

        if ( deltanm ) rtbnodes[relindx].hipdelta  )
                return (0);

        /* the version exists; return the pointer into the release list */
        return ( &rtbnodes[relindx] );
} /* end verifyrd */
```

```
        /* The version name argument was submitted. Perform error checks */
        /* and conversions to integer */

#include "defines.h"    /* constants */
#include "tablestr.h"   /* table structures */

struct relnode
*versubm( treeptr, vername, treetble, rtbnodes, relnum, deltanum )

        /* input arguments */
struct treentry *treeptr;       /* tree pointer */
char vername[]; /* version name */
struct treentry treetble[];     /* tree table */
struct relnode rtbnodes[];      /* release table */
        /* outout arguments */
int *relnum;    /* release number */
int *deltanum;  /* delta number */

        /* function value returned */
/* 'versubm' returns the pointer to the release's node in the release table */
/* if the version exists. Otherwise it returns a 0.*/

{
        char *redlmptr; /* pointer to the release delimiter in 'vername' */
        char *fndendch(); /* returns a pointer to a delimiter character */
        struct relnode *rdptr;  /* temporary release pointer */
        struct relnode *verifyrd();     /* release node pointer */
        int relindx;    /* index into the release table */

        /* See if the user submitted the -v 'r' option where r is the
           release number only.  If so, then convert the release string
           to integer and get its highest delta number.  */

        if ( redlmptr = fndendch( vername, '\0' ) )
        {
                *relnum = convatob( vername, redlmptr );
                        /* Verify that the release exists */
                if ( *relnum ) treeptr->numrels )
                {
                        orintf( "%d: no such release in tree %s\n",
                                *relnum, treeptr->trename);
                        return (0);
                }

                        /* The release number is valid.  Get its
                           index into the release table.  */
                relindx = getrindx( treeptr, rtbnodes, *relnum );
                        /* Get its highest delta */
                *deltanum = rtbnodes[relindx].hipdelta;
                return (&rtbnodes[relindx]);
        }
        /* 'fndendch' found a non-numeric embedded in the version
            name.  It might be the release delimiter, "." with a
           delta following.  */

        /* convert the version name to integer release and */
```

```
        /* delta numbers. 'INTRD' returns false if the */
        /* name is invalid. */

        if ( !(intrd (vername, relnum, deltanum) ) )
        {       printf( " Invalid release.delta number\n " );
                return (0);
        }
        else
        {       /* 'VERIFYRD' returns the pointer to the */
                /* version, if the version exists.      */

                if ( !( rdptr = verifyrd ( treeptr, treetble,
                        rtbnodes, *relnum, *deltanum ) ) )
                {       printf ( "%s   No such release.delta\n",
                                vername );
                        return (0);
                } /* end if */
         }
        return (rdptr);
}
```

```
/* Write out the change table, deleting the open version's entry */

#include (stdio.h)
#include "defines.h"
#include "tablestr.h"

wrtchgtb( fp_temp, chgtble, chgtbsz, chgptr )

FILE *fp_temp;  /* temporary new module file to which the table is written */
struct change chgtble[];        /* change table */
int chgtbsz;    /* change table size */
struct change *chgptr;  /* pointer to entry to be deleted */

{
        int numentry;   /* number of entries to be written out */
        struct change *ptr; /* traverses the change table */
                /* delete the change table entry */
        for ( ptr = chgptr + 1; ptr ( chgtble + numentry; ptr++, chgptr++ )
        {
                chgptr->time = ptr->time;
                chgptr->trenum = ptr->trenum;
                strcpy( chgptr->userid, ptr->userid );
        }

                /* Decrement the change table size and write it out */
        chgtbsz--;
        fwrite( &chgtbsz, sizeof(chgtbsz), 1, fp_temp);
                /* Write out the change table */
        fwrite( chgtble, sizeof( struct change ), chgtbsz, fp_temp );
}
```

```
/* Write out a control character and a control record */

#include (stdio.h)
#include "contlstr.h"

wrtcontl( contlchr, contlptr, fp_out )

        /* input arguments */
char contlchr;  /* control character */
struct control *contlptr;       /* pointer to the control record */
        /* output arguments */
FILE *fp_out;   /* output file */
{
        putc( contlchr, fp_out);
        fwrite(contlptr, sizeof(struct control), 1, fp_out);
}
```

APPENDIX F

TCS COMMAND SCRIPTS

When the TCS user issues a TCS command, he/she is actually executing a shell script whose name is of the form: <command>.com. An alias for each command is set up to invoke the the caommand's script. For example, in the command line: "get sample.1," "get" is an alias for $home/exec/get.com.

This appendix contains two sets of command files. The first set is intended for the TCS user. Copies of the user versions reside on the directory, $home/exec/tcs_com. The second set is intended for a programmer's use in testing and debugging TCS. Copies of these reside on the directory, $home/exec/testcom.

```
$ tcsexe
$ pwd
/a3/s1/rcs0074.exec
$ cd test_com
$ pwd
/a3/s1/rcs0074.exec/test_com
$
$ ls *.com
cancelc.com    createtree.com  printemp.com  printmod.com  show.com
creatmod.com   get.com         printhis.com  put.com       tcancel.com
$
$
$ tcsexe
$ pwd
/a3/s1/rcs0074/exec
$ cd tcs_com
$ pwd
/a3/s1/rcs0074/exec/tcs_com
$
$ ls *.com
cancelc.com    createtree.com  printhis.com  put.com       tcancel.com
creatmod.com   get.com         printmod.com  show.com
$
```

cancelc.com

```
: /bin sh
: ` Cancel a change reservation for a version in a TCS file
:
: ` USAGE:    cancelc  modname  tree name
:       modname  is the TCS module name and is required
:       tree name  is also required.  If the tree is the main tree,
:     then "0" is submitted for the tree name `
:
modname=$1
started=`pwd`
tmpdir=/tmp/tcs.$$
:
: ` Make sure that the TCS file exists
if [ ! -r $modname.t -o ! -f $modname.t ]
then
        echo "Cannot find TCS file: $modname"
        exit 1
fi
:
mkdir $tmpdir
mkdir $tmpdir/temp
:
cp $modname.t $tmpdir
cp $modname.t cancelc_save/$modname.t
cd $tmpdir
:
ar x $modname.t t.$modname
:
if $HOME/exec/cancelc.main $*
then
        cp temp/$modname t.$modname
        ar r $modname.t t.$modname
        cp $modname.t $started
fi
:
rm -rf $tmpdir
rm -rf $tmpdir/temp
cd $started
```

---

creatmod.com

```
: '/bin/sh
: 'Create a module'
:
: 'USAGE:  creatmod  modname
: '      where  modname  is the new TCS module name and is required '
:
: ' CREATMOD assumes that a file called  modname  exists
: ' on the current working directory.  This is original version submitted
: ' by the user and MUST contain at least one line for the system to '
: ' function correctly. '
:
: ' Make sure that the TCS file does not already exist '
mod=$1
if [ $# -lt 1 ]
then
        echo "Module name missing"
        exit 1
fi
if [ -r $mod.t -o -f $mod.t ]
then
        echo "TCS file, $mod, already exists"
fi
: '      Make sure that there is an original version on the current directory
if [ ' -r $mod -o ' -f $mod ]
then
        echo "File, $mod, is not on the current working directory "
        exit 1
fi
if $HOME/exec/creatmod.main $*
then
: '               Archive the new module and the history file '
        ar cr $mod.t  t.$mod h.$mod
fi
: '      remove any files that might have been created '
rm -f t.$mod
rm -f h.$mod
```

---

createe.com

```
: !/bin sh
: ` Create a tree in a TCS module
:
: `USAGE: createe  modname  parent tree name  parent version number `
: `       new tree name `
:
: `      All arguments are required. `
: `       modname  is the TCS module name `
: `       parent tree name  is the new tree's parent tree `
: `       parent version number  is the parent's release.delta version number `
:         new tree name  is the tree to be created `
:
modname=$1
started='pwd'
: `  Make sure that the TCS module exists `
if [ ! -r $modname.t -o  -d $modname.t ]
then
        echo "Cannot find TCS file: $modname"
        exit 1
fi
tmodir='tmo/tcs.$$
mkdir $tmpdir
mkdir $tmpdir/temp
:
cp $modname.t $tmpdir
:
: `Do all work in the temporary directory`
cd $tmpdir
ar x $modname.t t.$modname
:
if $HOME/exec/createe.main $*
then
        cp temp/$modname t.$modname
        ar r $modname.t t.$modname
        cp $modname.t $started/$modname.t
fi
:
rm -rf $tmpdir
cd $started
```

---

get.com

```
: '/bin/sh
:
modname=$1
if [ ' -r $modname.t -o ' -- $modname.t ]
then
        echo "Cannot find tcs file: $modname"
        exit 1
fi
:
tmpdir=/tmp/tcs.$$
:
mkdir $tmpdir
mkdir $tmpdir/temp
mkdir $tmpdir/diff
cp $modname.t $tmpdir
: ' save the current working directory name'
started='pwd'
: ' Do all work in the temporary directory '
cd $tmpdir
ar x $modname.t t.$modname n.$modname
: 'get the requested version'
if $HOME/exec/get.main $*
then
: '     If the change table has been updated, then archive the module file '
        if [ -r temp/tmpfile2 -o -f temp/tmpfile2 ]
        then
                cp temp/tmpfile2 t.$modname:
                ar r $modname.t t.$modname:
                cp $modname.t $started/$modname.t
        fi
        : ' deliver the file to the user '
        cp temp/$modname $started/t.$modname
fi
: ' Cleanup '
rm -rf $tmpdir
cd $started
```

---

printthis.com

```
: !/bin/sh
: `  Print the TCS module file (in a readable form, of course).`
: `  The output from this command is left in a file called:`
: `    module_name.s on the current working directory.`
:
: `  USAGE:  printthis  module_name  [optional_directory_name]`
: `       module_name  is required.`
: `       [optional_directory_name] is submitted if the TCS module`
: `                file is not on the current working directory.`
: `       If  optional_directory_name  is omitted, this command will`
: `       look for a file called "module_name.t" on the current`
: `       working directory.`
:
started=`pwd`
modname=$1
if [ $# -gt 1 ]
then
        pathname=$2/$modname.t
else
        pathname=$modname.t
fi
:
: `      Check that the TCS  file exists.`
if [ ! -r $pathname -o ! -f $pathname ]
then
        echo "Cannot find TCS file: $pathname"
        exit 1
fi
tmpdir=/tmp/printhis.$$
mkdir $tmpdir
cp $pathname $tmpdir/$modname.t
cd $tmpdir
ar x $modname.t h.$modname
$HOME/exec/printhis.main h.$modname   $started/$modname.s
rm -rf $tmpdir
cd $started
more $modname.s
```

printmod.com

```
: !/bin/sh
: ' Print the TCS module file (in a readable form, of course)'
: ' The output from this command is left in a file called:'
: ' <module name>.p on the current working directory'
:
: ' USAGE:  printmod <module name> <optional directory name>'
: '      <module name> is required.'
: '      <optional directory name> is submitted if the TCS module'
: '              file is not on the current working directory '
: '      If <optional directory name> is omitted, this command will '
: '      look for a file called "<module name>.t on the current '
: '      working directory.'
:
started=`pwd`
modname=$1
if [ $# -gt 1 ]
then
        pathname=$2/$modname.t
else
        pathname=$modname.t
fi
:
: '      Check that the TCS  file exists '
if [ ! -r $pathname -o ! -f $pathname ]
then
        echo "Cannot find TCS file: $pathname"
        exit 1
fi
tmpdir=/tmp/printmod.$$
mkdir $tmpdir
cp $pathname $tmpdir/$modname.t
cd $tmpdir
ar x $modname.t t.$modname
$HOME/exec/printmod.main t.$modname > $started/$modname.p
rm -rf $tmpdir
cd $started
more $modname.p
```

---

put.com

```
: !/bin/sh
: ' Put a new version into the TCS module file '
:
:
modname=$1
if [ ! -r $modname.t -o ! -f $modname.t ]
then
        echo "Cannot find tcs file: $modname"
        exit 1
fi
:
tmpdir=/tmp/tcs.$$
:
:
:
mkdir $tmpdir
mkdir $tmpdir/temp
mkdir $tmpdir/diff
cp $modname $tmpdir
cp $modname.t $tmpdir
: ' save the current working directory name'
started='pwd'
: ' Do all work in the temporary directory '
cd $tmpdir
ar x $modname.t t.$modname h.$modname
if $HOME/exec/put.main $*
then
: ' Figure out the differences between the old and new versions '
        if diff temp/$modname $modname > diff/$modname
        then
                /bin/echo "New version is identical to old version"
                /bin/echo "Delta was not created"
        else
                if $HOME/exec/putdelta.main $modname
                then
                        cp histmp'h.$modname
                        cp temp/$modname t.$modname
                        ar r $modname.t t.$modname h.$modname;
                        cp $modname.t $started/$modname.t
                fi
        fi
fi
: ' Cleanup '
: ' Get back to the user's directory '
rm -rf $tmpdir
cd $started
```

show.com

```
: !/bin/sh
:
modname=$1
if [ ! -r $modname.t -o ! -f $modname.t ]
then
        echo "Cannot find tcs file: $modname"
        exit 1
fi
:
tmpdir=/tmp/tcs.$$
:
mkdir $tmpdir
cp $modname.t $tmpdir
: ' save the current working directory name'
started='pwd'
: ' Do all work in the temporary directory '
cd $tmpdir
ar x $modname.t t.$modname h.$modname
$HOME/exec/show.main $*
: ' Cleanup '
rm -rf $tmpdir
cd $started
```

```
:cancel.com

: /bin/sh
: ' Cancel a change reservation for a version in a TCS file '
:
: ' USAGE:    cancelc <modname> <tree name> '
: '      <modname> is the TCS module name and is required '
: '      <tree name> is also required.  If the tree is the main tree, '
: '      then "0" is submitted for the tree name '
:
modname=$1
started='pwd'
tmpdir=/tmp/tcs.$$
:
: ' Make sure that the TCS file exists '
if [ ! -r $modname.t -o ! -f $modname.t ]
then
        echo "Cannot find TCS file: $modname"
        exit 1
fi
:
mkdir $tmpdir
mkdir $tmpdir/temp
:
cp $modname.t $tmpdir
cd $tmpdir
:
ar x $modname.t t.$modname
:
if $HOME/exec/:cancel.main $*
then
        cp temp/$modname t.$modname
        ar r $modname.t t.$modname
        cp $modname.t $started
fi
:
rm -rf $tmpdir
cd $started
```

cancelc.com

```
: /bin/sh
: ` Cancel a change reservation for a version in a TCS file
:
: ` USAGE:    cancelc  modname   tree name
: `       modname  is the TCS module name and is required
: `       tree name  is also required.  If the tree is the main tree,
:       then "0" is submitted for the tree name `
:
modname=$1
started=`pwd`
tmpdir=/tmp/tcs.rce
:
: ` Make sure that the TCS file exists `
if [ ! -r $modname.t -o ! -f $modname.t ]
then
        echo "Cannot find TCS file: $modname"
        exit 1
fi
:
: `      remove any temporary directories that might be hanging around `
rm -rf $tmpdir
rm -rf $tmpdir/temp
mkdir $tmpdir
mkdir $tmpdir/temp
:
cp $modname.t $tmpdir
cp $modname.t cancelc_save/$modname.t
cd $tmpdir
:
ar x $modname.t t.$modname
:
if $HOME/exec/cancelc.main $*
then
        cp temp/$modname t.$modname
        ar r $modname.t t.$modname
        cp $modname.t $started
fi
:
cd $started
```

---

creatmod.com

```
: /bin/sh
: `Create a module`
:
: `USAGE:  creatmod modname
: `      where  modname  is the new TCS module name and is required
:
: ` CREATMOD assumes that a file called  modname  exists
: ` on the user and MUST contain at least one line for the system to `
:   function correctly.
:
: ` Make sure that the TCS file does not already exist `
mod=$1
if [ $# -lt 1 ]
then
        echo "Module name missing"
        exit 1
fi
if [ -r $mod.t -o -f $mod.t ]
then
        echo "TCS file, $mod, already exists"
fi
: `      Make sure that there is an original version on the current directory`
if [ ! -r $mod -o ! -f $mod ]
then
        echo "File, $mod, is not on the current working directory "
        exit 1
fi
if $HOME/exec/creatmod.main $*
then
: `              Archive the new module and the history file `
        ar cr $mod.t  t.$mod h.$mod
fi
: `      remove any files that might have been created `
rm -f t.$mod
rm -f h.$mod
```

---

createe.com

```
: '/bin/sh
: ' Create a tree in a TCS module
:
: 'USAGE: createe  modname  parent tree name  parent version number
: '       new tree name '
:
: '      All arguments are required. '
: '       modname  is the TCS module name '
: '       parent tree name  is the new tree's parent tree '
: '       parent version number  is the parent's release.delta version number
: '       new tree name  is the tree to be created '
:
modname=$1
started='pwd'
: '   Make sure that the TCS module exists '
if [ ! -r $modname.t -o ! -f $modname.t ]
then
        echo "Cannot find TCS file: $modname"
        exit 1
fi
: '   FOR TESTING. save the TCS module '
cp $modname.t cr_treesave
: '       Make temporary directories for intermediary files '
: tmpdir=/tmp/tcs.$$
 tmpdir=/tmp/tcs.rcs
rm -rf $tmpdir
mkdir $tmpdir
mkdir $tmpdir/temp
:
cp $modname.t $tmpdir
:
: 'Do all work in the temporary directory'
cd $tmpdir
ar x $modname.t t.$modname
:
if $HOME/exec/createe.main $*
then
        cp temp/$modname t.$modname
        ar r $modname.t t.$modname
        cp $modname.t $started/$modname.t
fi
:
: ' Get back to the user's directory '
cd $started
: 'remove not done for testing'
```

---

get.com

```
: !/bin/sh
:
modname=$1
if [ ! -r $modname.t -o ! -f $modname.t ]
then
        echo "Cannot find tcs file: $modname"
        exit 1
fi
:
: use known temp directory for testing
: tmpdir=/tmp/tcs.$$
tmpdir=/tmp/tcs.rcs
:
: ' remove any temporary versions that might be hanging around '
rm -rf $tmpdir
:
mkdir $tmpdir
mkdir $tmpdir/temp
mkdir $tmpdir/diff
cp $modname.t $tmpdir
: ' For testing, save the old version '
cp $modname.t getsave $modname.t
: ' save the current working directory name'
started='pwd'
: ' Do all work in the temporary directory '
cd $tmpdir
ar x $modname.t t.$modname h.$modname
: ' Delete any temporary versions that might be hanging around '
: 'get the requested version'
if $HOME/exec/get.main $*
then
: '     If the change table has been updated, then archive the module file '
        if [ -r temp/tmpfile2 -o -f temp/tmpfile2 ]
        then
                cp temp/tmpfile2 t.$modname;
                ar r $modname.t t.$modname;
                cp $modname.t $started/$modname.t
        fi
        : ' deliver the file to the user '
        cp temp/$modname $started/t.$modname
fi
: ' Cleanup '
: ' Get back to the user's directory '
cd $started
```

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

printemo.com

: '/bin/sh

:   Print the temporary line numbered version of the TCS file created on '
: ' by the last "out" operation.  '
:
: ' This command is used for debugging and assumes that the testing version'
: ' of put.com was run, thus saving the line numbered file in a "putsave" '
: ' directory.
: ' The output is saved in a file called  modname.r  on the '
: ' current working directory '
:
: ' USAGE:  printemo  module name   optional directory name '
: '       module name  is required.'
: '       optional directory name  is submitted if the "putsave" directory '
:       is not a subdirectory of the current working directory.  '
: '       If  optional directory name  is omitted, this command will '
:       look for a file called "putsave/ module name .l" on the current '
: '       working directory.'
:
modname=$1
if [ $# -gt 1 ]
then
        pathname=$2/putsave/$modname.l
else
        pathname=putsave/$modname.l
fi
:
: '      Check that the TCS  file exists '
if [ ! -r $pathname -o ! -f $pathname ]
then
        echo "Cannot find TCS file: $pathname"
        exit 1
fi
:
$HOME/exec/printemp.main $pathname   $modname.r
cat $modname.r
-------------------------------------------------------
```

---

printhis.com

```
:  /bin/sh
:   Print the TCS module file in a readable form.  Of course
:   The output from this command is left in a file called:
: `  module_name.s on the current working directory.`
:
: ` USAGE:  printhis  module_name  optional_directory_name `
: `       module_name  is required.`
: `       optional_directory_name  is submitted if the TCS module`
:                 file is not on the current working directory `
: `      If  optional_directory_name  is omitted, this command will `
: `      look for a file called "module_name.t on the current `
: `      working directory.`
:
started=`pwd`
modname=$1
if [ $# -gt 1 ]
then
        pathname=$2/$modname.t
else
        pathname=$modname.t
fi
:
: `      Check that the TCS  file exists `
if [ ! -r $pathname -o ! -f $pathname ]
then
        echo "Cannot find TCS file: $pathname"
        exit 1
fi
tmpdir=/tmp/printhis.rcs
mkdir $tmpdir
cp $pathname $tmpdir/$modname.t
cd $tmpdir
ar x $modname.t h.$modname
$HOME/exec/printhis.main h.$modname > $started/$modname.s
rm -rf $tmpdir
cd $started
cat $modname.s
```

---

printmod.com

```
: `/bin/sh
: ` Print the TCS module file in a readable form. Of course
: ` the output from this command is left in a file called:`
: `  module name .p on the current working directory`
:
: ` USAGE:  printmod  module name  optional directory name `
: `       module name  is required.`
: `       optional directory name  is submitted if the TCS module`
: `             file is not on the current working directory `
: `       If optional directory name is omitted, this command will `
: `       look for a file called "$module name.t on the current `
: `       working directory.`
:
started=`pwd`
modname=$1
if [ $# -gt 1 ]
then
        pathname=$2/$modname.t
else
        pathname=$modname.t
fi
:
: `      Check that the TCS  file exists `
if [ ! -r $pathname -o ! -f $pathname ]
then
        echo "Cannot find TCS file: $pathname"
        exit 1
fi
tmpdir=/tmp/printmod.rcs
mkdir $tmpdir
cp $pathname $tmpdir/$modname.t
cd $tmpdir
ar x $modname.t t.$modname
$HOME/exec/printmod.main t.$modname > $started/$modname.p
rm -rf $tmpdir
cd $started
cat $modname.p
```

put.com

```
: /bin/sh
: ' Put a new version into the TCS module file '
:
:
modname=$1
if [ ! -r $modname.t -o  -f $modname.t ]
then
        echo "Cannot find tcs file: $modname"
        exit 1
fi
:
: use known temp directory for testing
: tmpdir=/tmp/tcs.$$
tmpdir=/tmp/tcs.rcs
:
: ' remove any temporary versions that might be hanging around'
:
rm -rf $tmpdir
:
mkdir $tmodir
mkdir $tmodir/temp
mkdir $tmpdir/diff
co $modname $tmpdir
cp $modname.t $tmpdir
: ' For testing, save the old version '
cp $modname.t putsave $modname.t
cp $modname putsave $modname
: ' save the current working directory name'
started='pwd'
: ' Do all work in the temporary directory '
cd $tmpdir
ar x $modname.t t.$modname h.$modname
if $HOME/exec/put.main $*
then
: ' FOR TESTING.  save the line numbered version of the TCS file'
        cp temp/tmpfile2 $started/putsave/$modname.l:
else
        cd $started:
        exit 1
fi
: ' Figure out the differences between the old and new versions '
if diff temp/$modname $modname   diff/$modname
then
        /bin/echo "No differences found between the old and new version'
        /bin/echo "Detla was not created"
        cd $started:
        exit 1
fi
if $HOME/exec/putdelta.main $modname
then
        cp histmp h.$modname
        cp temp/$modname t.$modname
        ar r $modname.t t.$modname h.$modname:
        cp $modname.t $started/$modname.t
fi
: ' Cleanup '
: ' Get back to the user's directory '
cd $started
```

---

show.com

```
: /bin/sh
:
modname=$1
if [ ! -r $modname.t -o ! -f $modname.t ]
then
        echo "Cannot find tcs file: $modname"
        exit 1
fi
:
: use known temp directory for testing
: tmpdir=/tmp/tcs.$$
tmpdir=/tmp/tcs.rcs
:
: ' remove any temporary versions that might be hanging around
rm -rf $tmpdir
:
mkdir $tmpdir
cp $modname.t $tmpdir
:   save the current working directory name`
started=`pwd`
: ' Do all work in the temporary directory '
cd $tmpdir
ar x $modname.t t.$modname h.$modname
$HOME/exec/show.main $1
: ' Cleanup '
: ' Get back to the user's directory '
cd $started
```

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

xcancel.com

: /bin/sh
: ' Cancel a change reservation for a version in a TCS file
:
: ' USAGE:     cancelc .modname   tree name
: '       modname  is the TCS module name and is required
: '       tree name  is also required.  If the tree is the main tree, '
: '      then "0" is submitted for the tree name '
:
modname=$1
started='pwd'
tmpdir=/tmp/tcs.rcs
:
: ' Make sure that the TCS file exists
if [ ! -r $modname.t -o ! -f $modname.t ]
then
        echo "Cannot find TCS file: $modname"
        exit 1
fi
:
: '      remove any temporary directories that might be hanging around '
rm -rf $tmpdir
rm -rf $tmpdir/temp
mkdir $tmpdir
mkdir $tmpdir temp
:
cp $modname.t $tmpdir
cp $modname.t cancelc_save/$modname.t
cd $tmpdir
:
ar x $modname.t t.$modname
:
if $HOME/exec/xcancel.main $*
then
        cp temp/$modname t.$modname
        ar r $modname.t t.$modname
        cp $modname.t $started
fi
:
cd $started
-------------------------------------------------------
-------------------------------------------------------
```

APPENDIX G

COMPILE AND LINK SCRIPTS

G.1 COMPILING TCS .C FILES

Compiling one or more .c files should be done on the $home/source directory, using one of two shell scripts:

tcsall.cmp

cmpl

The script, tcsall.cmp, compiles all files listed in the file tcs_souurce.list (isuue an "ls .c > tcs_source.list" before running tcsall.cmp). The script "cmpl" is used to compile one file. Both scripts leave the object code on the $home/obj directory.

COMPILE AND LINK SCRIPTS

A copy of the scripts, tcsall.cmp and cmpl, follows:

```
$ pwd
/a2/s1/rcs0074/source
$
$ cat tcsall.cmo
: ' USAGE      tcsall.cmp < tcs_source.list '
: ' compile all .c files in the TCS system '
: '  NOTE: file names are in tcs_source.list '
:
while read filename
do
        if [ x$filename = x ]
        then
                exit 0
        fi
        cc -c $filename
        mv $filename.o $HOME/obj/$filename.o
        echo `ls -l $HOME/obj/$filename.o`
done
$
$
$ cat cmpl
# Compile a c function and move the object file to the 'obj' directory
#
# usage: cmpl <basename>
#
cc -c $1.c;
mv $1.o /a2/s1/rcs0074/obj/$1.o
$
```

G.2 LINKING TCS MODULES

Scripts for linking all TCS modules are on the $home/obj directory. The alias: "obj" is set up for changing to this directory.

COMPILE AND LINK SCRIPTS

Each TCS module (one for each TCS command) is linked using the script: <module_name>.link where <module_name> is the same as the TCS command that invokes the executable. All link scripts produce an executable module whose file names is of the form: <module_name>.main, on the $home/exec directory.

There are two additional link scripts:

userintr.olink

tcs.link

The script, userintr.olink, produces an object file made up of functions from the "user interface group." The object is left on the $home/obj directory to be used in subsequent links. The script, tcs.link, links are modules in the TCS systems by invoking userintr.olink and then the scripts for each module.

A listing of all link scripts follows:

```
$ cds $home/obj
$ ls $home/obj/*.*link
/a3/s1/rcs0074/obj/cancel.link      /a3/s1/rcs0074/obj/put.link
/a3/s1/rcs0074/obj/creatmod.link    /a3/s1/rcs0074/obj/putdelta.link
/a3/s1/rcs0074/obj/creatree.link    /a3/s1/rcs0074/obj/show.link
/a3/s1/rcs0074/obj/get.link         /a3/s1/rcs0074/obj/tcs.link
/a3/s1/rcs0074/obj/printemp.link    /a3/s1/rcs0074/obj/userintr.olink
/a3/s1/rcs0074/obj/printhis.link    /a3/s1/rcs0074/obj/xcancel.link
/a3/s1/rcs0074/obj/printmod.link
$
```

```
$
$ cat usenintr.olink
ld   -r      parcopta.o gopt.o tcsopt.o util.o  fndtrdnm.o gethird.o versubm.o g
etrindx.o getreptr.o intrd.o verifyrd.o fndendch.o convatob.o -lc;
mv a.out userintr.o
$
$  ____________________                                    ____
$
$ obj
$
$ cat tcs.link
# Link all modules in the TCS system
#
#       USERINTR is used in other links and must be done first
echo "linking USERINTR"
userintr.olink
echo "linking GET"
get.link
echo "linking PUT"
put.link
echo "linking PUTDELTA"
putdelta.link
echo "linking CREATMOD"
creatmod.link
echo "linking CREATREE"
creatree.link
echo "linking PRINTMOD"
printmod.link
echo "linking PRINTEMP"
printemp.link
echo "linking CANCELC"
cancelc.link
echo "linking XCANCEL"
xcancel.link
echo "linking PRINTHIS"
printhis.link
echo "linking SHOW"
show.link
```

COMPILE AND LINK SCRIPTS

```
$ cancelc.link
$ cat cancelc.link
# Link 'cancelc'and put the executable on the 'exec' directory
#
cc -o /a2/sl/rcs0074/exec/cancelc.main  cancelc.o readtble.o getreptr.o chlopen.
o    wrtrtble.o wrtchgtb.o rwbody.o
$
$
$ cat creatmod.link
# Link 'creatmod' and move the executable to the 'exec' directory
#
cc -o /a2/sl/rcs0074/exec/creatmod.main creatmod.o writabls.o creathis.o getcomm
t.o updcomtb.o
$
$
$ cat creatree.link
# Link the creatree module
cc -o /a2/sl/rcs0074/exec/creatree.main creatree.o readtble.o writabls.o userint
r.o updtrtbl.o rwbody.o chkrsvc.o
$
$
$ cat get.link
cc -o  /a2/sl/rcs0074/exec/get.main   get.o  readtble.o writabls.o chlopen.o gpt
hname.o gethis.o gcomind.o userintr.o hldintst.o getversn.o setflag.o queue.o
$
$
$ cat printmod.link
# Link the 'printmod' executable and leave the output on the 'exec' dir.
#
cc -o /a2/sl/rcs0074/exec/printmod.main printmod.o readtble.o printbl.o
$
$
```

```
$ cat printhis.link
cc -o /a2/s1/rcs0074/exec/printhis.main printhis.o;
$
$
$ cat printemp.link
# Link the 'printemp' executable and leave the output on the 'exec' dir.
#
cc -o /a2/s1/rcs0074/exec/printemp.main printemp.o readtble.o printbl.o
$
$
$ cat put.link
# link 'put'
#
cc -o /a2/s1/rcs0074/exec/put.main  put.o openall.o closeall.o readtble.o getrep
tr.o chkopen.o getcommt.o updtbls.o updhist.o wrtrtble.o wrtrhgtb.o hldintsl.o g
etversn.o updcomtb.o rwhistf.o queue.o setflag.o
$
$
$ cat show.link
# Link the 'SHOW' module
#
cc -o $home/exec/show.main show.o userintr.o showtree.o streeopt.o prtrinfo.o pr
trel.o showchil.o readtble.o gpthname.o chkopen.o
$
$
$ cat xcancel.link
# Link 'xcancel'and put the executable on the 'exec' directory
#
cc -o /a2/s1/rcs0074/exec/xcancel.main  xcancel.o readtble.o getreptr.o chkopen.
o    wrtrtble.o wrtrhgtb.o rwhody.o
```