A CONFIGURATION ITEM AND BASELINE IDENTIFICATION SYSTEM FOR SOFTWARE CONFIGURATION MANAGEMENT

by

WILLIAM H. WILSON, IV.

B.S., Virginia Polytechnic Institute and State University, 1970
M.B.A. University of North Carolina at Greensboro, 1983

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Approved by:

Major Professor

Table of Contents

LIST OF FIGURES

Chapter 1 - Overview

1.1 Introduction

This paper addresses the subject of Software Configuration Management (SCM) and presents a master's project which consists of the design, development, and implementation of a SCM identification system at Kansas State University. Software configuration management is the use of configuration management to manage software systems (2, 3). Configuration management has been defined as "... the discipline of identifying the configuration of a system at discrete points in time for the purpose of maintaining the integrity and traceability of the configuration throughout the life cycle." (2).

The SCM identification system was designed and developed to be part of the prototype software development environment under development at Kansas State University. The KSU prototype development environment is designed to be used in the development of fifth-generation software and in the Department of Defense's STARS environment (10, 12).

This report presents a detailed review of the current literature on SCM (with a special emphasis on literature which describes the implementation of SCM tools), followed by a presentation of the SCM identification project which has been implemented at Kansas State University. A summary of the project and a discussion of its relationship to current SCM thought are presented in this chapter, the project's requirements are presented in Chapter 2, the design in Chapter

3, implementation details in Chapter 4, and a summary of the project and possible extensions are presented in Chapter 5.

The term "tool" is used throughout this paper to refer to a software system which has been developed to assist individuals or other software systems perform a specific task in the software development life cycle.

1.2 Literature Review - Overview

Most of the current literature on SCM can be divided into two categories: literature which addresses SCM in a tutorial, theoretical manner (i.e., material which explores the nature of SCM) and literature which documents the implementation of actual SCM tools. A review of literature in each category is presented.

1.3 Tutorial, Theoretical SCM Literature

Tutorial, theoretical literature is composed of literature which provides an instructional overview of SCM and literature which addresses certain aspects of SCM from an instructional or theoretical viewpoint.

Much of the current overview material on SCM has been written by Messrs. E. H. Bersoff and S. G. Siegel individually, together and with others (2, 3, 4, 6, 7). Typically, this material explores the concepts of software and software configuration management (SCM), and presents the components of SCM. Each of these concepts and the components of SCM will be addressed in this section.

1.3.1 The Concept of Software

In the past, software has been viewed as actual computer programs composed of coded statements. Recently, this view has been expanded to include just about anything that contributes to eventual written software code (4). Bersoff, Henderson and Siegel have defined software to be "information" which is (4):

(1) structured with logical and functional properties;
(2) created and maintained in various forms and representations during its life cycle; and
(3) tailored for machine processing in its fully developed state.

This definition leads to the idea that software can exist in two forms: nonexecutable form and executable form (3, 4). Nonexecutable software includes specifications, documentation, and software stored on a machine readable storage media but which is not executable (e.g., source code). Executable software is the actual machine executable code. From a SCM perspective, both forms of software are viewed in the same manner (4). Therefore, for SCM purposes, the term software includes specifications, documentation, source code, and executable code.

1.3.2 Software Configuration Management

Bersoff, Henderson, and Siegel define configuration management as "the discipline of identifying the configuration of a system at discrete points in time for the purpose of systematically controlling changes to this configuration and maintaining the integrity and traceability of the configuration throughout the system life cycle" (3). A similar but more

specific definition is used by the Department of Defense. The DOD regulation on configuration management defines CM as "A discipline applying technical and administration direction and surveillance to (1) identify and document the functional and physical characteristics of a configuration item [to be defined later]; (2) control changes to those characteristics; and (3) record and report change processing and implementation status" (8). In both cases, CM is viewed as the discipline for identifying configuration elements, controlling changes to the configuration, and monitoring the status of the configuration. Traditionally, CM has been used in the management of hardware systems to insure consistent labeling of hardware elements, to track the evolution of hardware components throughout their life, and to control changes to hardware components (4).

Software configuration management is CM adapted to systems composed primarily of software (3). Thus, SCM is concerned with identifying software components, tracking the evolution of software configurations, and controlling changes to the software configuration throughout the life cycle of the configuration. Bersoff, Henderson, and Siegel state that "The primary objective of SCM is the effective management of a software system's life cycle and its evolving configuration" (4).

Interest in the application of CM to software systems has grown out of the numerous software failures which occurred in the 1970's (2). Such failures are generally attributed to a lack of control over the software development process. The goals of SCM along with the other quality

assurance disciplines (quality assurance, validation and verification, and test and evaluation) are to control the software development process and to aid in the development of software systems which exhibit product integrity (4). A product that exhibits integrity is one that "fulfills user needs, can easily and completely be traced through its life cycle, meets specified performance criteria, and whose cost and delivery expectations are met" (3).

In addition to the concept of product integrity, other concepts which are fundamental to SCM are the concepts of software configuration, baseline, and configuration item.

A software configuration is a "relative arrangement of software parts" (3). Typically, a software configuration is a software system which is composed of specifications, documentation, and programs which are related in some way (e.g., part of the same production computer system). Thus, a configuration can be thought of as the collection of all current software parts for a given system.

A baseline (BL) is a "reference point or plateau in the development of a system" (4). Bersoff refers to it as a "snapshot" of the software configuration at a given point in time (2). A baseline, like a configuration, is a collection of related software elements. A baseline differs in that it is a picture of the configuration as it exited at a given point in time. Bersoff, Henderson, and Siegel use the term baseline to identify the approved (final) product associated with each stage in the life cycle of a system (3). They define functional

baseline as being associated with the end of the system concept formulation stage, allocated baseline as being associated with the end of the advanced development/validation stage, design baseline as being associated with the end of the detailed design phase, product baseline as being associated with the end of the first article development stage, and operational baseline as being associated with the end of the production/deployment stage (3).

A configuration item (CI) is the "most elementary" entity in the software configuration (2). The software components which can be called CIs can be many different things. Typically, a software configuration item is thought of as source code for a program. However, this is a rather limited view of what a CI can be. From a SCM perspective, a CI can be any item in the software development process which is considered important enough to name. A configuration item can represent an entire specification, a piece of the specification, a data structure, a program, etc. Bersoff, Henderson, and Siegel describe CIs as being related to one another in a "tree-like hierarchy" (2, 3). They envision that a CI can be partitioned into multiple CIs as a system evolves during its life cycle.

In retrospect, a baseline can be defined as "ordered collection" of CIs and a software configuration as a collection of related CIs (3). A baseline is the set of CIs at specified update levels formally included in the baseline. A configuration is the total set of CIs and associated updates that have been identified in the configuration. That is, a

baseline is a picture of a system's configuration at a given point in time, and a configuration is the current view of all that has been identified within the system.

1.3.3 SCM Tasks

SCM is composed of four components or tasks: identification, control, auditing, and status accounting. A discussion of each task is presented.

1.3.3.1 SCM Identification

The identification component of SCM is concerned with the consistent identification of CIs, baselines, changes to CIs, and changes to baselines. This is perhaps the most important SCM task. Without consistent identification of CIs, baselines and changes to each, it would be impossible to perform the other SCM tasks.

CIs must be consistently and uniquely labeled. Each CI must have a unique name and a mechanism must exist that can be used to differentiate between different versions of the same CI. Each CI must also have a description associated with it which describes the CI, when it was created and by whom. Similarly, each CI version must have a description which describes its uniqueness compared to other versions of the CI, when it was created, and by whom. All of this identification information must be recorded and maintained (see status accounting later in this chapter).

As noted earlier, Bersoff, Henderson, and Siegel have suggested a tree-like hierarchy of relationships for CIs in which a CI can be progressively decomposed into a series of more detailed CIs (3). To keep track of CIs, they have suggested a naming convention in which the first CI is identified as CI 1. The next level of decomposition for this CI would yield CIs 1,1; 1,2; 1,3; etc. The next level below 1,1 would be 1,1,1; 1,1,2; etc.

Equally important to the SCM identification task is the baseline identification function. Baselines must be consistently labeled with unique names. As with CIs, a mechanism must exist which can be used to differentiate between different updates or versions of a baseline. Thus, baseline identification involves labeling of baselines and updates to baselines. A description must be associated with the baseline and with each update, and all information regarding a baseline must be recorded and maintained.

1.3.3.2 SCM Control

The focus of the SCM control task is on managing changes to CIs and baselines (4).

Bersoff, Henderson, and Siegel state that "The role of software configuration control is to provide the administrative mechanism for precipitating, evaluating, and approving or disapproving all change proposals throughout the system life cycle" (4). They list three "base ingredients" for software configuration control (4):

1) Documentation (such as administrative forms and supporting technical and administrative material) for formally precipitating and defining a proposed change to a software system.
2) An organizational body for formally evaluating and approving or disapproving a proposed change to a software system (the Configuration Control Board).
3) Procedures for controlling changes to a software system.

The first ingredient can be addressed by a computerized modification request system. The second ingredient is by necessity a manual administrative process. The third ingredient can be addressed with a computerized CI and baseline identification and control system.

Bersoff has suggested that a software control system should include a centralized repository for software components (CIs), facilities for developing and maintaining CIs, control of software releases, automatic program and documentation reconstruction, and automatic change tracking and reporting (2).

1.3.3.3 SCM Auditing

The auditing component of SCM is concerned with how the current state of a system fulfills the requirements for the system. The SCM auditing component is composed of verification and validation functions. Verification is concerned with verifying that the intent of a CI in a previous baseline or baseline update is carried forward to the next baseline or baseline update (2, 4). Validation is concerned with ensuring that the software configuration addresses and solves the problem that it was intended to solve (2, 4). That is, it makes sure that the needs of the user are satisfied.

Bersoff presents SCM auditing as the means to formerly establish a baseline (2). Accordingly, each to-be-established baseline in the system life cycle must be audited before it can be labeled an established (approved) baseline.

The auditing task is primarily a manual management task. However, data provided by the status accounting task (next section) can be used in the performance of the auditing task.

1.3.3.4 SCM Status Accounting

The SCM status accounting component is concerned with the tracking and reporting of all activities associated with CIs and baselines (3). Its function is to maintain a record of how a configuration has evolved over its life cycle. SCM status accounting responsibilities include tracking all changes to a configuration from the time they are requested until the time they are implemented or rejected, tracking the implementation of new and changed software, tracking all documentation associated with a configuration, and keeping track of which modules have been implemented and where they have been implemented (8). The SCM status accounting component is responsible for recording the occurrence of configuration events, when they occurred, what they entailed (i.e., what happened), and who initiated the event (3). Configuration events include creation of CIs, the initiation of a request to change or add to the configuration, CI updates, etc. With all of this data, the status accounting function will be able to provide a description of where a configuration is at any given point in time.

SCM status accounting provides the data to support the other three SCM components. In particular, it provides the data needed by the SCM auditing function.

1.4 Implemented SCM Tools

A variety of development tools which include SCM functions have been developed and implemented. The more notable tools are discussed in this section.

1.4.1 The Source Control System

Rochkind has described the Source Code Control System (14). The Source Code Control System (SCCS) is a SCM identification and control tool for the management of text data. It is currently distributed as part of the UNIX* operating system. SCCS is basically a tool which can be used to identify software modules (i.e., software configuration items) and control the changes which are made to these modules. SCCS provides facilities for uniquely identifying modules, storing modules, retrieving modules, updating modules, uniquely identifying module versions (modification levels), preventing parallel modifications to a module, controlling access to modules, and recording change data such as date, time, description, and changed by whom.

---

* Trademark of AT&T Bell Laboratories.

Modules are identified in SCCS by a name and a SCCS identification string (SID). The name uniquely identifies a module while the SID uniquely identifies the version of that module. For example, suppose that a module is given the name "moda" when created. The initial version of "moda" will be 1.1, the second version will be 1.2, the third 1.3, and so on. Version as used here indicates a particular state of a module which differs from the previous state and succeeding state in some way. Therefore, version 1.2 is version 1.1 with modifications. The format of a SID is (16):

release.level.branch.sequence

Release and level (e.g., 1.2) comprise the normal SID identification for a module and are referred to as being on the "trunk" (i.e., the main version line) of a module. Release, level, branch, and sequence (e.g., 1.2.1.1) are used to identify a "branch" from the trunk of a module. For example, one may decide to modify version 1.2 of a module after version 1.3 has been created. In such a situation, one would cause a branch to be taken from version 1.2 of the module. The SID for this branch will be 1.2.1.1. An update to "moda" 1.2.1.1 will be identified via SID 1.2.1.2. Thus, release and level are used together for SIDs on the trunk of module, and release, level, branch, and sequence are used together on a branch from the trunk of a module. The entire sequence of versions, including those on the trunk and those on branches, for a module is referred to as a "tree".

An important function performed by SCCS is change control. SCCS will not allow two login names to retrieve a module for update at the same

time. Thus, the possibility of parallel updates to a module is avoided. SCCS also requires the specification of a change description before a module can be updated (i.e., a new version is created).

SCCS stores all pertinent information regarding a module in one file. The original source for the module is stored along with module description, change information, change description, etc. Revisions or versions of a module are stored as deltas, where a delta is the difference between two successive versions of a module. Any version of a module can be retrieved by specifying the version desired when entering the SCCS "get" command. SCCS will then perform a forward merge of deltas to the original source until the desired version is obtained.

SCCS provides the mechanism for controlling update access privileges for each module. The SCCS file for each module has a list, embedded within the file, of UNIX login names which are allowed to make changes to that module. The designated SCCS administrator for a project is the only one allowed to modify this list. An additional access control measure is also imposed by the UNIX operating system in conjunction with SCCS. Login names outside the group level of the designated SCCS administrator (the UNIX owner of the SCCS files for a project) are not allowed read or write access to the SCCS files for that project unless they are specifically added to the access list of the designated files by the administrator.

SCCS functions are initiated via a set of SCCS commands. These commands are normally entered by a terminal user. However, SCCS commands could

be initiated from "C" programs via the UNIX "fork" and "execv" system calls.

1.4.2 The Revision Control System

Tichy has described the Revision Control System (15). The Revision Control System (RCS) is a SCM identification and control system for the management of text data such as source code and documentation. RCS is similar to SCCS in function and form. Like SCCS, RCS is a tool which can be used to identify text modules and control changes to these modules. It has been implemented in a UNIX environment as is SCCS. Capabilities provided by RCS are the ability to uniquely identify text modules, store modules, retrieve modules, update modules, uniquely identify updates to modules, prevent parallel updates to a module, and control access to modules.

RCS uses a similar module identification scheme to that used by SCCS. Modules are identified by a unique name and a revision number. The RCS revision number has a somewhat different format from that of SCCS. RCS supports the "branching" concept used by SCCS to create a second version line off of the trunk of a module. RCS also supports "branches" off of branches, which is not supported by SCCS. While not specified by Tichy (15), the structure of the RCS revision number appears to be:

release.level.branch.sequence.branch.sequence

RCS has the facility to assign a symbolic name to a given revision or a branch in a module's tree. SCCS does not have a similar facility.

RCS maintains a change history for each module within the source file for that module. Updates are not allowed without the specification of an update description. RCS also prevents the retrieval of a module for update by more than one user, thus preventing parallel updates.

All information regarding a module is stored within the module as is done by SCCS. The major differences between RCS and SCCS are the manner in which the information is stored and the way a specific version is generated. RCS stores the most up-to-date version of a module along with reverse separate deltas for all previous versions of the module. SCCS stores the initial version of a module and forward merged deltas for successive versions of the module. The result is that RCS works backwards to get to old versions of a module while SCCS works forward to get to newer versions of a module.

In summary, RCS and SCS are very similar products with the major differences being the way deltas are stored and the way module versions are generated. Both appear to provide the essential ingredients required for the SCM identification function, and both provide SCM control facilities.

1.4.3 The Project Automated Librarian

The Project Automated Librarian (PAL) has been described by Prager (13). PAL is a SCM identification and control tool developed by IBM for use in the management of medium sized software development projects. It can be used to manage the development of software code. PAL provides

facilities for the creation, storing, retrieval, modification, and listing of source modules.

PAL is similar to SCCS and RCS in providing the aforementioned facilities. Other similarities include: the unique identification of files (modules in SCCS and RCS), the prevention of parallel updates to a file, the maintenance of change history for a file, and the capability to retrieve earlier versions of a file. Additionally, PAL requires a description of the change when a file is updated.

PAL differs from SCCS and RCS in a number of ways: PAL is software code oriented instead of just text oriented; PAL maintains complete copies of previous versions of a file instead of deltas to a base file; PAL provides no access security; and PAL is menu driven instead of command driven. Additionally, PAL stores information regarding dependency relationships. An update to a file will automatically regenerate other files which are derived in some way from the file which has just been updated.

1.4.4 The GTD-5 EAX Development Environment

Begley has presented an overview of the GTD-5 EAX software development environment (1). This environment is a development environment which is composed of a formalized software development methodology, a number of development support tools, and a distributed hardware and operating system environment. The support tools include several SCM tools: the Software Management Support System, load generation tools, and a change

tracking system.

Begley indicates that the Software Management Support System (SMSS) is based on a customized use of UNIX SCCS to "create" a software module, to "fetch" a module and to "promote" a module (i.e., update a module). He provides no details on how the SMSS implementation differs from the native SCCS implementation. The load generation tools are tools used to assemble a complete load package for a specific GTD-5 EAX processor. The change tracking system is the software system which is used to formally request new software features or changes to existing software. Requests are accepted from a computer terminal. Request status and assigned responsibilities are maintained and reported periodically.

1.4.5 The Modification Request Control System

The Modification Request Control System (MRCS) has been described by Knudsen, Barofsky, and Satz (9). MRCS is a SCM change control tool developed to manage modification requests. A modification request as used here is a request to modify (i.e., add , change or delete) a software system.

MRCS interfaces with users interactively. The format of a modification request is left to the user for a given implementation of MRCS. The system operates on a mode or status basis, where the status of a modification request changes from "submitted" to other states during the life cycle of the modification request. The status of a modification request can be displayed at any time. The system may be used to record

modification request for documentation, source code, etc.

MRCS was designed to be used in conjunction with SCCS and other software to form a change management system. Change management is defined as "the function that seeks to insure that when a product (or new release of a product) is issued, it contains only (and all) those modifications which the producer and consumer expect it to contain." (9). Knudsen, Barofsky, and Satz maintain that use of MRCS with SCCS, software which maintains a configuration list of change levels or versions of software per release, and software which will generate a specific release of a system will constitute use of a change management system.

1.5 Literature Summary

As stated at the beginning of this chapter, most of the current literature relating to SCM focuses on tutorials of SCM and on implemented SCM tools.

The tutorial material, in general, attempts to define SCM and to place it in proper perspective within the software development environment. This material discusses the four tasks of SCM (identification, control, auditing, and status accounting) in some detail. Much of the material on identification centers around the form needed to uniquely identify CIs. Bersoff, Henderson, and Siegel have suggested a hierarchical relationship between CIs, with the implication that CIs can spawn other CIs (3).

There appears to be general agreement on what SCM is, its objectives, and its tasks (7). The differences or unclear issues are associated with what to identify (documents as a whole, chapters of documents, etc.) and how to identify CIs. Bersoff, Henderson, and Siegel suggest a numeric decomposition labeling scheme, while Rochkind and Tichy have implemented basic identification systems based on unique names and numeric version numbers.

The literature on implemented tools addresses identification and control aspects of SCM with an emphasis on software which can be used to identify and control updates to software components, and manage change requests. Rochkind with SCCS and Tichy with RCS have presented two basic tools which can be used to identify configuration items and to control changes to CIs in a UNIX environment. Prager has presented a SCM system for the IBM environment which goes beyond the CI identification stage by actually defining dependencies between components. Begley has presented an overview of a CI identification tool which appears to use SCCS in a customized way along with a modification request system. Knudsen, Barofsky, and Satz have presented a UNIX based modification request system which was designed to work with SCCS and other software to form what they characterize as a change control system.

1.6 Summary of the Configuration and Baseline Identification Project As It Relates to the Literature

The objective of this project is to develop a system which can be used to identify, control, and track CIs and baselines for requirement specifications, and which will be flexible enough to use throughout the system life cycle with any text oriented development data (e.g., design specifications, data descriptions, source code, etc.). Implementation of this system will be based on a customized usage of SCCS as is suggested by Begley (1). The system will provide the capability to explicitly identify baselines in addition to CIs. Baseline identification has not been addressed by any of the tools which were examined. The system will provide two of the four parts which Knudsen, Barofsky, and Satz state are required for a change management system (9): a source identification and control capability (based on SCCS) which allows the regeneration of previous versions of a software item, and a baseline identification and control capability which keeps track of CIs (by version) which are included in each baseline. A future interface to a modification request system and use of software which can regenerate any release of a system would result in a change management system per their definition.

Chapter 2 - Requirements for the SCM Identification Project

2.1 Introduction

The objective of this project is to design, develop and implement a SCM identification system which is composed of functions which can be used to identify, control and track configuration items (CIs) and baselines (BLs). The initial use of this system will be for requirement specifications.

The SCM identification system will be incorporated into the prototype development environment currently under development at Kansas State University (KSU). The SCM identification system will be used by the other KSU development environment tools (requirements specification tool, design specification tool, etc.) to store and retrieve text data. Figure 2-1 provides a graphical representation of the SCM system and its envisioned place in the KSU prototype software development environment.

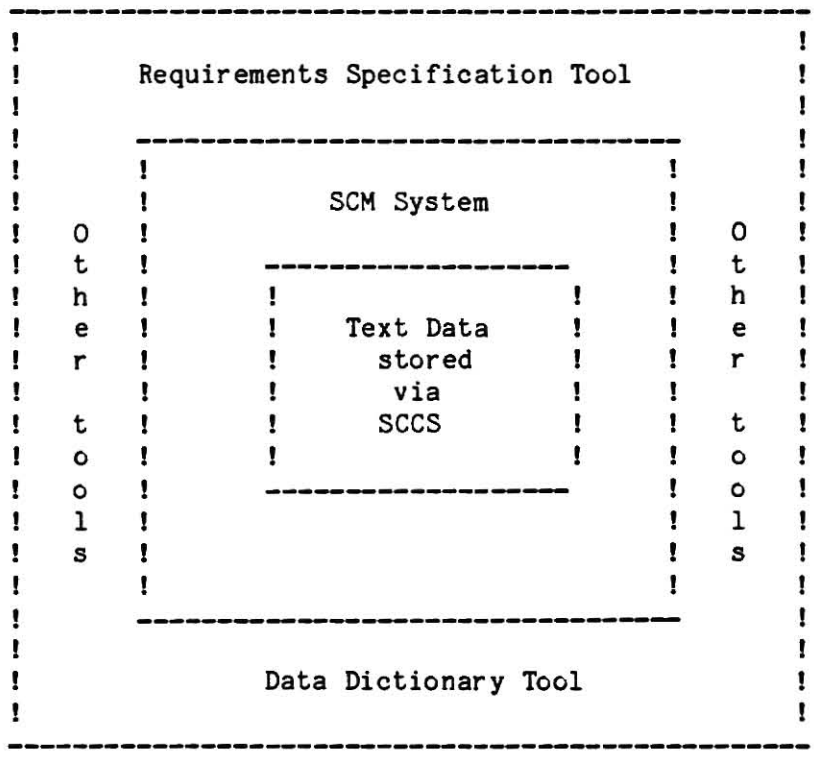

Figure 2-1. Envisioned Role of the SCM System

2.2 General Requirements

The SCM identification system must be suitable for use in a standard UNIX environment and must be implemented in the Kansas State University UNIX environment. All SCM identification functions (presented later in this chapter) must be implemented using "C" language and/or shell programs.

In general, the SCM identification System must support the identification, storage, and retrieval of configuration items, and the identification, storage and retrieval of baselines.

The SCM identification system must be general enough to be used with any text oriented development data (i.e., it must have a wide range of applicability). Therefore, the format of CIs can not be specified by the SCM identification system.

The system must have simple and easy to use external interfaces in order to promote its use in the development process. Interfaces must be provided for UNIX login sessions, shell programs, and "C" language programs.

To support simple external interfaces, input data must be command and argument list oriented. Output must be simple and pertinent to the user's request and needs. Input and output should not consist of elaborate crt screens. To do so would make the use of the system by a shell or "C" program cumbersome at the least.

CI activities must be user oriented. Access controls by individual CI must be provided. The system must also prevent parallel updating of CIs. The contents of a CI must contain the text data provided by the user along with other SCCS data (such as the list of authorized users for that CI).

Baseline activities must be restricted to the designated project administrator for each project. Baseline data must be stored in a SCCS baseline file associated with the subject project file. The contents of the text of a baseline will contain a list of CI names and their SIDs.

Administrative functions must be provided in order to establish and maintain the appropriate directories and SCCS files required for a project by the SCM identification system. Administrative activities must be restricted to the designated project administrator for a given project.

2.3 Specific Requirements

The identification requirements can be subdivided into administrative, CI identification, and baseline identification requirements. Common to both CI and BL identification are the requirements to be able to create, retrieve, update and list entities (CIs and BLs). Support for the incremental development of CIs and BLs, automatic archiving of CIs and BLs, the selective retrieval of CIs and BLs by version, and the prevention of parallel updates to CIs and BLs are also required.

2.3.1 Administrative Requirements

Administrative requirements include the ability to install the SCM identification system and the ability to add projects to an installed SCM identification system. Use of these functions must be restricted to the designated administrator for each project. Specific functions required are:

(1) Install the SCM Identification Subsystem (install): The install

function will install the SCM identification system for a new project administrator by creating a SCM global directory under the file system directory under which the administrator's login directory is located, and a SCM project ID directory under the SCM directory. Additional project IDs can be added to a previously installed SCM system via the addproj function which is described later. Individual project files (e.g., source, data, etc.) will have to be added via the addfile function which is also described later. Input to the function will consist of a project ID, the system administrator's login name, and the file system identifier. Output will consist of a SCM directory, a project ID directory and various messages.

(2) Add a project to a previously installed SCM system (addproj): The addproj function will add a project directory to an installed SCM system by creating a project ID directory under the SCM directory of the caller. Input will consist of a project ID. Output will be a project directory and various messages to the user.

(3) Add a project file directory (addfile): The addfile function will add project files (e.g., source, data, etc.) to a previously established project (see addproj) of an installed SCM system (see install) by creating a project file under the specified project directory and creating a baseline directory under the project file directory. Input will consist of a project ID and a project file name. Output will consist of a project file directory and a

baseline directory for the subject project.

(4) Update the list of authorized users (updatusr): The updatusr function will add or remove login names from the list of authorized users for a specified CI or for all CIs in a specified project file by causing the list of authorized users stored in each CI to be updated. Input will consist of a project ID, a project file name, a CI name, and an update type flag followed by a list of login names. Output will consist of an updated SCCS file and various messages.

2.3.2 CI Identification Requirements

As previously noted, identification requirements include the ability to create, retrieve, update, and list CIs. CI activities are end user oriented. Specific functions required are:

(1) Create a CI (createci): The createci function will cause the creation of a CI by creating a member (SCCS file) for the CI under the specified project file. The function will require the specification of a CI "description" and provide the option to specify a CI "type" (e.g., "activity", "data", etc.). The calling tool may require that a CI "type" be provided at all times which would make "type" a requirement to all users of that calling tool. Input to this function will consist of a project ID, project file name, CI name, CI type (optional), and a CI description. Output will consist of a SCCS file and various messages.

(2) Fetch a CI (fetchci): The fetchci function will retrieve a specific version (SCCS SID) of a CI for information use or for subsequent update. The latest version will be fetched if a specific version is not requested. The option must be provided to retrieve a CI for information only or for subsequent update. The fetchci function must not allow the retrieval of a CI for update if that CI has already been fetched for update nor allow the retrieval of a CI for update from an unauthorized user. Input will consist of a project ID, a project file name, a CI name, and a fetch type flag. Output will consist of a copy of the CI in the directory of the requester from which the fetchci was issued and various messages.

(3) Update a CI (updateci): The updateci function will create a new version of a CI which has been previously fetched for update by causing the SCCS file for the subject CI to be updated. The function will require a "description" of the new version before update is allowed and one or more modification request numbers if a modification request system is being used for the subject project. The updateci function will not allow anyone but the user who fetched a CI for update to update that CI. Input will consist of a project ID, a project file name, and a CI name. Output will consist of the updated SCCS file for the subject CI and various messages.

(4) List information regarding one or more CIs (listci): The listci function will list specific information pertaining to a CI.

Options will be provided to list the description of a CI, its change history or both at the latest version level or at a specified version level. Input will consist of a project ID, a project file name, a CI name, and an options flag. Output will consist of a standard output listing of the requested information and various messages.

(5) Replace a CI's description (rcidesc): The rcidesc function will replace a CI's description by causing the description of a CI which is stored in the SCCS file for the CI to be replaced. Input will consist of a project ID, a project file name, a CI name, and a file which contains the new description. Output will consist of the updated SCCS file for the specified CI.

(6) Amend (add to) the description of a CI version (civdesc): The civdesc function will add to the description of the specified version of a CI by appending the new description onto the old description which is stored in the SCCS file for the specified CI. The option will be provided to specify the amendment on the command line or to receive a terminal prompt to key in the description. Input will consist of a project ID, a project file name, a CI name, a CI version number, and the amendment. Output will consist of the updated SCCS file for the specified CI and various messages.

2.3.3 Baseline Identification Requirements

Baseline identification requirements include the ability to create, retrieve, update, and list BLs. Baseline activities must be restricted to the designated project administrator for each project. Specific functions required are:

(1) Create a baseline (createbl): The createbl function will create a new baseline from the latest version of each CI in the configuration. The function will require the specification of a file in which the baseline "description" can be found. The function will also support the options to include selected CIs by version and to exclude selected CIs. Only one version of a CI will be allowed in a baseline. The use of this function will be restricted to the project administrator. Input from the administrator will consist of a project ID, a project file name, and a description. Input from the CI portion of the system will consist of CI files from which CI names and SIDs will be extracted. An optional CI include/exclude file can also be specified as input. Output will consist of a SCCS file for the new baseline and various messages.

(2) Fetch a baseline (fetchbl): The fetchbl function will retrieve a specific version of a baseline for information purposes or for subsequent update. The fetchbl function will restrict the fetch for update to the designated project administrator. Fetchbl will also prevent the retrieval for update of a baseline version if that

version has already been retrieved for update. Input will consist of a project ID, a project file name, and a baseline version number. Output will consist of a copy of the BL in the directory of the requester from which the fetchbl was issued and various messages.

(3) Update a baseline (updatebl): The updatebl function will update the baseline version which was previously fetched for update. The use of this function will be restricted to the project administrator. The function will require a "description" of the update before the update is allowed. Input will consist of a project ID, a project file name, and a description. Output will consist of an updated SCCS file for the subject BL and various messages.

(4) List a baseline (listbl): The listbl function will list information pertaining to a baseline. Options will be provided to list the change history of a baseline or the description of a baseline. Input will consist of a project ID, a project file name, a baseline version, and an options flag. Output will consist of a standard output listing of the requested information and various messages.

(5) Replace a baseline's description (rbldesc): The rbldesc function will replace a BL's description. The use of this function will be restricted to the project administrator. Input will consist of a project ID, a project file name, a baseline version number, and the name of the file in which the description can be found. Output will consist of an updated SCCS file for the specified baseline.

(6) Amend (add to) the description of a baseline version (blvdesc): The blvdesc function will add to the description of a specified baseline version. The option will be provided to specify the amendment on the command line or to receive a terminal prompt to key the description at the terminal. The use of this function will be restricted to the project administrator. Input will consist of a project ID, a project file name, a baseline version number, and the amendment. Output will consist of the updated SCCS file for the specified BL and various messages.

2.4 Design Decisions

The requirements for this project restrict its development and operations to the UNIX environment. Requirements for the support of multiple CI and baseline versions, no parallel updating, and support for update access controls influence the use of UNIX Source Code Control System (SCCS) functions as primitive functions for the SCM identification system.

The many functional requirements that have been defined tend to influence the implementation toward one or more "C" programs. Implementation via shell programs would require a like number of programs to perform all the required functions. The many functional requirements also imply a modular structure which can be used in a "C" program.

Chapter 3 - Design for the SCM Identification Functions

The SCM identification system has been designed as a hierarchy of functional modules. Each function described under the specific requirements section of Chapter 2 will constitute a separate functional module in the actual system. Each functional module will be designed to be independent of all other modules with the exception of the supervisor module. A supervisor module will be used to interpret the caller's request and to call the appropriate functional module. Each module will in turn utilize specific SCCS functions and/or other UNIX facilities to accomplish the required task. Figures 3-1 through 3-4 provide hierarchy diagrams of the system's module structure.

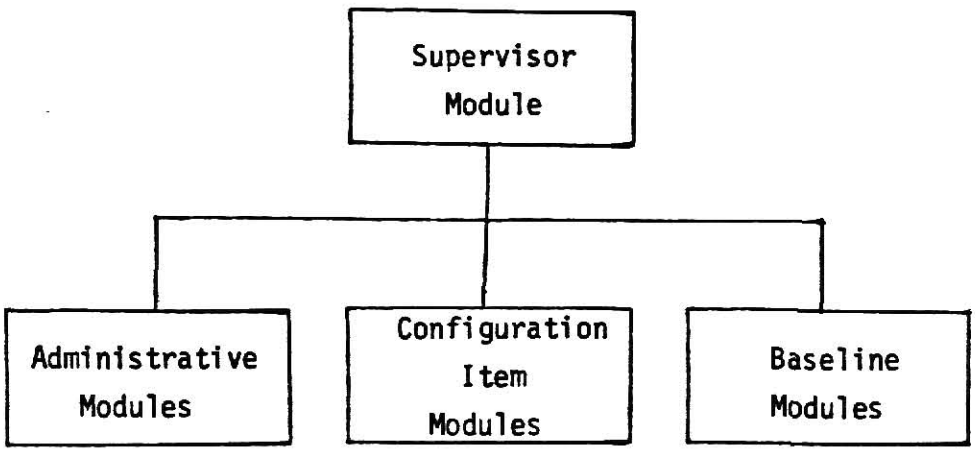

Figure 3-1. SCM High Level Hierarchy

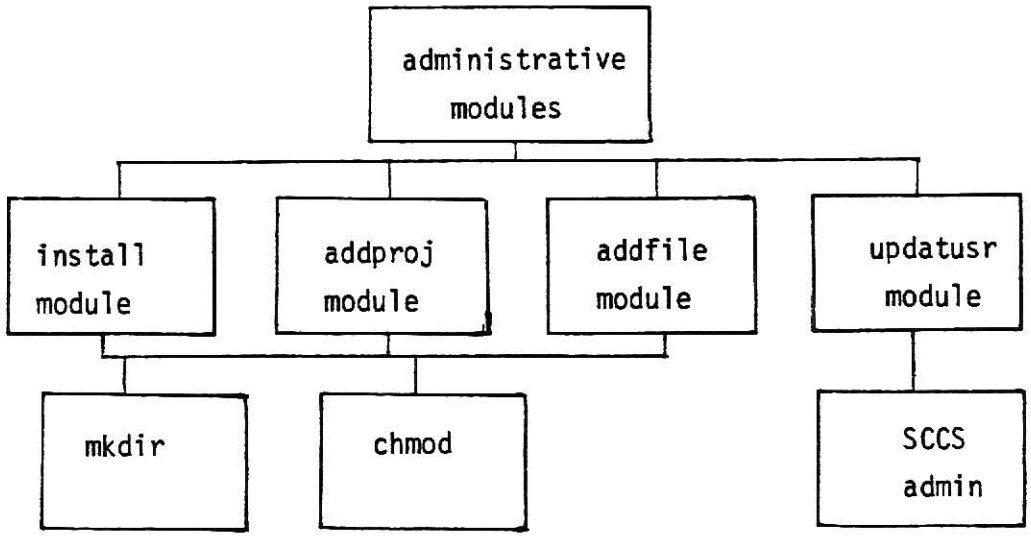

Figure 3-2. SCM Administrative Module Hierarchy

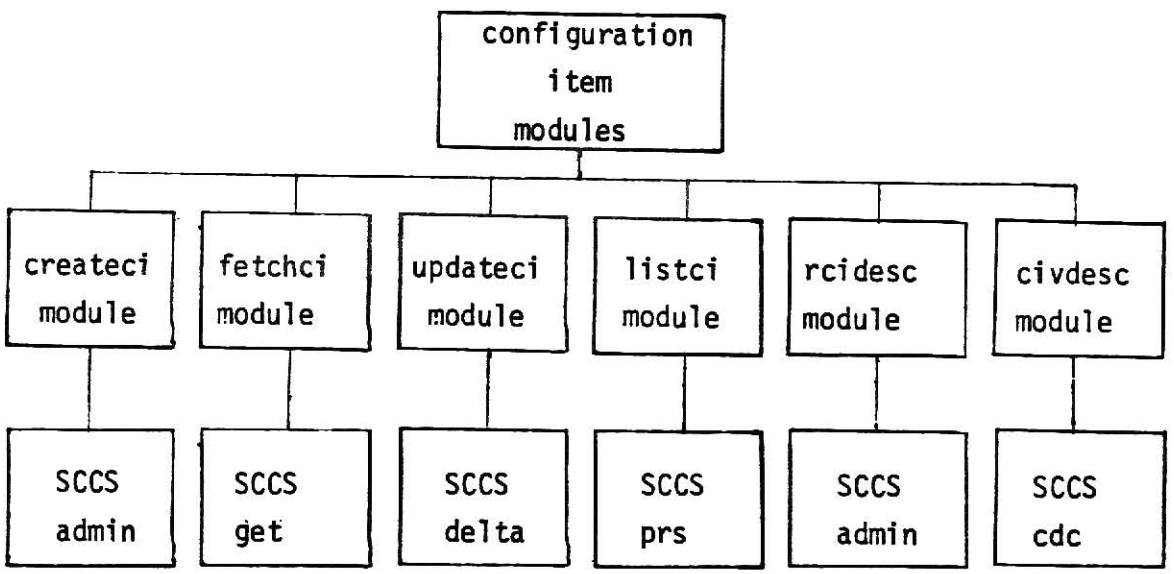

Figure 3-3. SCM Configuration Item Module Hierarchy

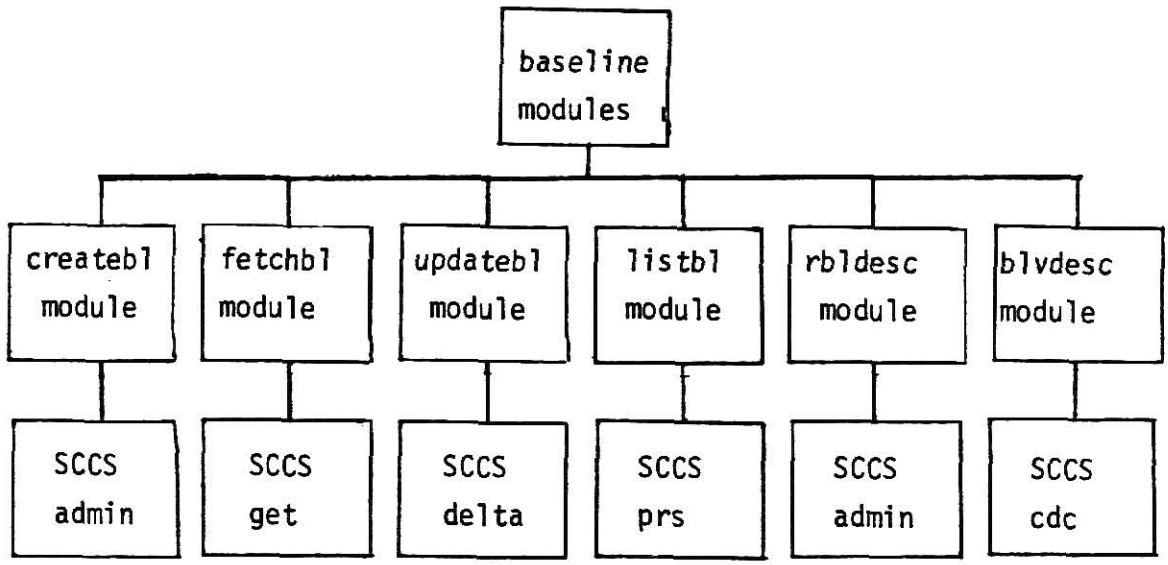

Figure 3-4. SCM Baseline Module Hierarchy

Input to the supervisor will consist of a command which is composed of a list of arguments. The first argument will identify the function to be performed and the remaining arguments will be functionally dependent. Figure 3-5 provides a graphical representation of the data structure of the command. Standard input for each functional module will consist of a series of arguments which include the project ID and project file name as well as other functionally dependent arguments.

This design will promote good module independence. The modules will be highly functional and each module will be interconnected, via data coupling, with only the supervisor module.

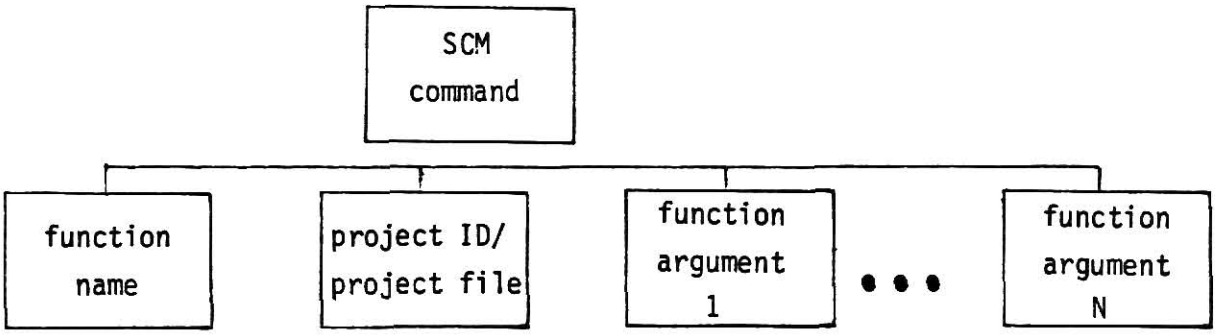

Figure 3-5. Command Data Structure

3.1 Administrative Modules

The administrative modules will perform the administrative functions such as installation of the SCM identification system, adding projects to a SCM identification system, and adding project files (e.g., requirements specification, source, data, etc.) under a SCM identified project. Each module will be designed to fulfill the requirements of the identically named function specified in Chapter 2.

The "install" module will create a SCM directory under the file system directory under which the administrator's login directory is located as is depicted in Figure 3-6. The "addproj" module will add a project under the administrator's SCM directory as is depicted in Figure 3-7.

The "addfile" module will add a SCM file under the specified project directory as is depicted in Figure 3-8. The "updatusr" module will allow the administrator to control update access privileges for each CI created under a file directory. Module specifications for each module are provided in Appendix A.

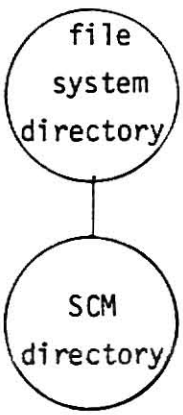

Figure 3-6. SCM Directory Created by install

Figure 3-7. Project ID Directory Created by addproj

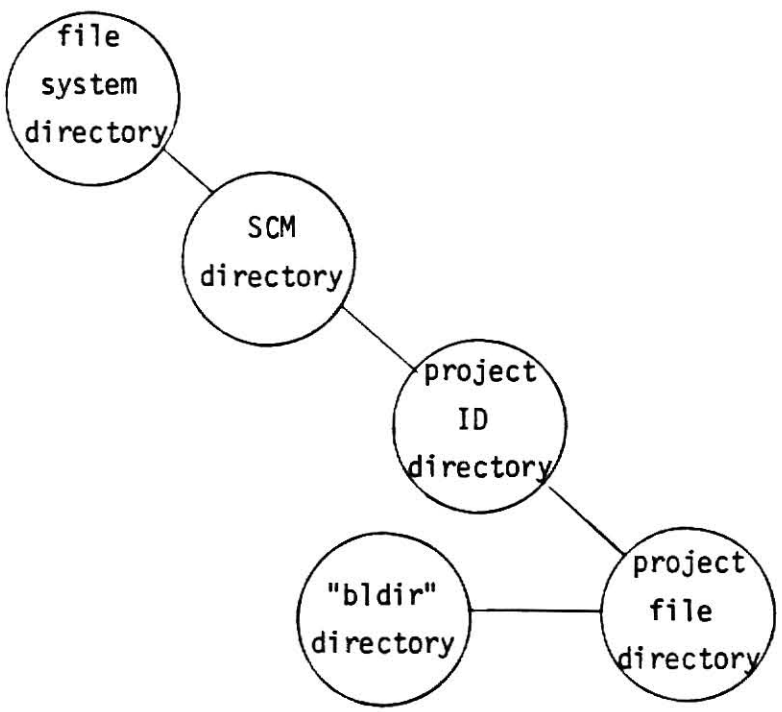

Figure 3-8. Project File Directory Created by addfile

3.2 CI Identification Modules

The CI modules will perform the creation, maintenance, and reporting functions for configuration items. Each module will be designed to fulfill the requirements of the identically named function specified in Chapter 2. Input to the CI modules will consist of the set of character string arguments passed by the supervisor and in some cases free form text files. Each CI is itself a free form text file. Output from the CI modules will be composed of CI files put in the requester's directory or in a SCCS file, and text to stdout and stderr.

The "createci" function will create a SCCS file for the new CI under the specified project ID and project file directories as is depicted in Figure 3-9. It will also require as input a free form text file which describes the CI. The "fetchci" CI will retrieve a module for information use or for subsequent update and place the CI in the current working directory of the requester. The CI will be retrieved from its SCCS file. The "updateci" module will update the SCCS file for the specified CI using the text file in the requester's current working directory (which has the same name as the CI) as the update text. It will also require a description of the update which is also free form text. The "listci" module will list the specified CI as requested in the argument list. The CI's SCCS file will be read as input. The "rcidesc" module will replace the description for a CI with the free form text which is provided in the specified description file. The "civdesc" module will amend the comments specified for a specific

version (release) of a CI. Comments are stored as free form text in the SCCS file for the CI. Module specifications for each module are provided in Appendix B.

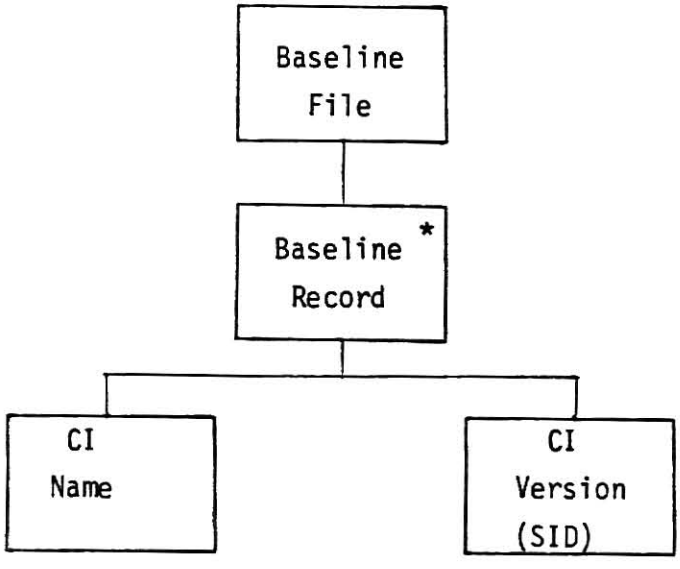

Figure 3-9. CI Creation by createci

3.3 Baseline Identification Modules

The baseline identification modules will perform the creation, maintenance, and reporting functions for baselines. Each module will be designed to fulfill the requirements of the identically named function specified in Chapter 2. Input to the baseline modules will consist of

the set of character string arguments passed by the supervisor, free form text in some cases, and in some cases formated records. Each baseline file is composed of a set of CIs and their versions (SIDs) separated from each other by a new line character (see Figure 3-10), and is stored in a SCCS file. Output from the baseline modules will be composed of baseline files put in the requester's directory or in a SCCS file, and text to stdout and stderr.

The "createbl" module will create a SCCS file for a new baseline and place it under the baseline directory for the specified project file as depicted in Figure 3-11. It will also require as input a free form text file which describes the baseline. The "fetchbl" module will retrieve a baseline for information use or for subsequent update and place it in the current directory of the requester. The baseline will be retrieved from its SCCS file. The "updatebl" module will update the SCCS file for the specified baseline using the baseline file in the requester's current working directory which has the same name. A description will be required in free form text format. The "listbl" module will list the specified baseline as requested via the arguments. The baseline SCCS file will be read as input. The "rbldesc" module will replace the description for a CI with the free form text which is provided in the specified description file. The "blvdesc" module will amend the comments specified for a specific version (release) of a baseline. Module specifications are provided in Appendix C.

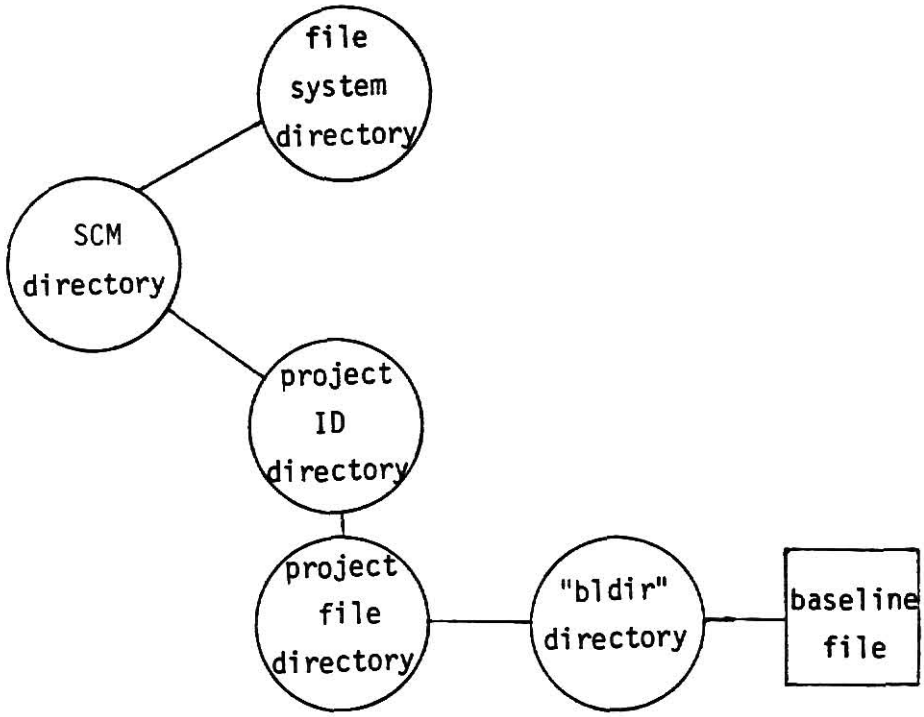

Figure 3-10. Baseline File Data Structure

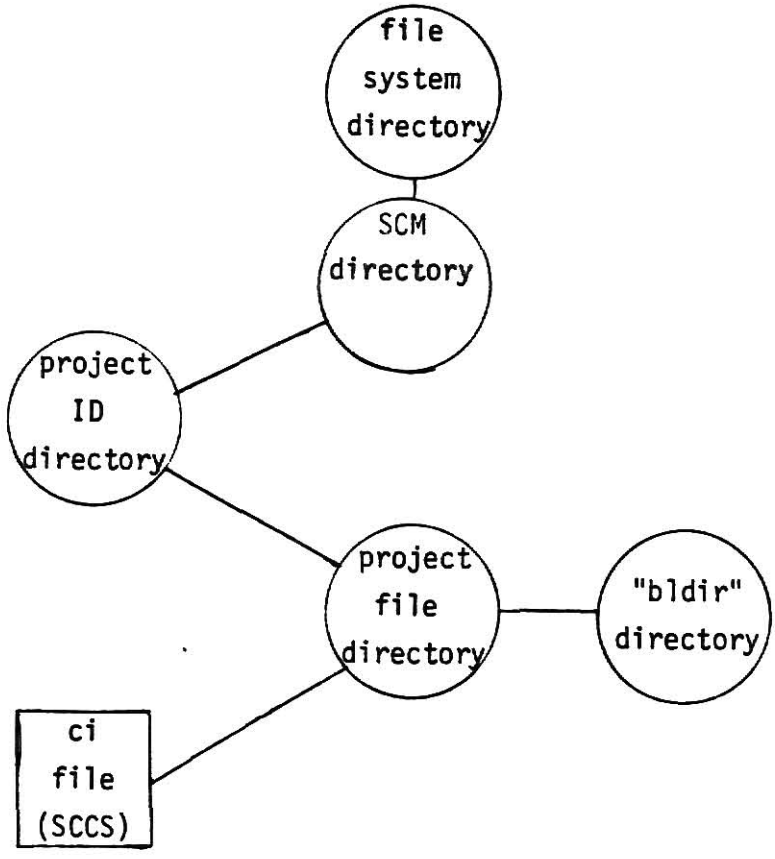

Figure 3-11. Baseline Creation by createbl

Chapter 4 - Implementation Details

The SCM identification system has been implemented as a single "C" program which is composed of many internal functions. The name of this program is "scmid". The "main" function of scmid is the supervisor module which is described in Chapter 3. Each of the other modules discussed in Chapter 3 are internal functions within scmid. The program is installed in such a way that it may be initiated by specifying the name of the desired SCM function and its argument list.

The "C" program is approximately 1700 statements long. A listing is provided in Appendix E.

4.1 Internal Interfaces

The "main" (supervisor) function of scmid will examine the first argument passed to it and determine which SCM function has been requested. The first argument passed to a "C" program always contains the name that the subject program was called by (i.e., the name that was used to execute the program). In the case of scmid, the first argument will contain the desired SCM function name.

4.2 External Interfaces

A user of the SCM identification system can call scmid in one of three ways: a "C" program call to scmid, a UNIX login ID call to scmid, or a shell program call to scmid.

A "C" program call to scmid requires that the caller create a child process via a UNIX "fork" system call and then overlay the child process with scmid via

execv ("/filesys/bin/scmcmd",argptrlist)

where *scmcmd* is the name of the SCM function desired and argptrlist is an array of pointers to the arguments required by the desired SCM function.

A call from a UNIX login to scmid requires the entry of

scmcmd arglist

where scmcmd is the name of the desired SCM function and arglist is the list of arguments required by the specified SCM function.

A call from a shell program to scmid requires the entry of

scmcmd arglist

or

nohup scmcmd arglist

where scmcmd is the name of the desired SCM function and arglist is the list of arguments required by the specified SCM function.

Most of the SCM functions in scmid use one or more of the SCCS functions as primitive functions. SCCS functions are called from scmid by first creating a child process via the UNIX "fork" system call and then overlaying the child process with the desired SCCS function via

execv ("/usr/bin/sccscmd",argptrlist)

where sccscmd is the name of the desired SCCS function and argptrlist is an array of pointers to the arguments required by the SCCS function.

4.3 Implementation Procedure

The SCM identification system can be implemented in a UNIX environment which supports SCCS. The system has been designed to be implemented for each UNIX file system. A SCM administrator should be designated for each file system in which the SCM system is to be used. This person must be the person under whose login name the CI and Baseline Identification System is implemented via the following procedure:

1. Move scmid.c to /filesys/bin where "filesys" is the file system identifier under which the administrator's login directory is located.

2. Position the current login to /filesys/bin (e.g., cd /filesys/bin)

3. Compile the program by entering

   ```
   cc scmid.c -oscmid
   ```

4. Turn on the "set user ID on" bit by entering

   ```
   chmod 4755 scmid
   ```

   This will allow all authorized users of the SCM system in the subject file system to inherit the privileges of the project administrator for the duration of each SCM request.

5. Define link names for the scmid program. Once this is done, a request to execute a SCM function will cause scmid to be executed. Link names are defined as follows:

   ```
   ln scmid install
   ln scmid addproj
   ```

```
ln scmid addfile
ln scmid updatusr
ln scmid createci
ln scmid fetchci
ln scmid updateci
ln scmid listci
ln scmid rcidesc
ln scmid civdesc
ln scmid createbl
ln scmid fetchbl
ln scmid updatebl
ln scmid listbl
ln scmid rbldesc
ln scmid blvdesc
```

6. Enter the SCM command "install" to install the system in the SCM administrator's UNIX file system.

Upon completion of the above procedure, the CI and Baseline Identification System will be installed in the subject UNIX file system as is described in the administrative module section of Chapter 3. The owner of the system will be the login name under which the "install" command was executed.

The "scmid" program can be modified to have an operational scope broader than a file system by altering the setting of "home" in "main" of scmid to something other than the current file system. If the system is so modified, then the above implementation instructions must be modified accordingly.

Chapter 5 - Conclusions and Extensions

This paper has presented a master's project in Software Configuration Management. A literature survey was presented first, followed by the requirements, design, and implementation of the subject project.

5.1 Conclusions

All the stated requirements for the Configuration Item and Baseline Identification System have been met. The system can be used to uniquely identify configuration items and baselines, to control changes to each, and to uniquely identify such changes. Furthermore, the system can be used to control and track the evolution of CIs and baselines throughout the life cycle of a configuration.

The CI and Baseline Identification System offers great flexibility in terms of what a CI can be and what baselines can signify. A CI can be any text oriented data from a specification document or piece thereof, to source code for a module or function. A baseline can be the end product of a system life cycle stage or a specific release of a system.

Inherent in the system design is the capability to keep like things together. Project separation (i.e., a separate project ID directory for each project) and project file separation (i.e., separate project file directories within each project ID) are part of the design.

Central control of the SCM identification process is included in the design. The creation of projects and project files as well as control

over CI access privileges and baseline functions are all restricted to a designated SCM project administrator.

5.2 Extensions

Several possible extensions to the CI and Baseline Identification System have been identified. A description of each follows:

- Interface this system with a modification request system. A key element of such an interface should be the validation of modification request numbers through the standard SCCS facility for validation of such numbers. Other potential benefits are the cross-referencing of CIs and modification request numbers, and relating satisfied modification requests with baselines.

- Make the specification that modification request numbers are required to update CIs a "project level" specification instead of a "CI level" specification.

- Require the specification of modification numbers at "fetch for update" time instead of at "update" time. This would discourage the retrieval of a CI for update unless a modification request number has been previously submitted which requires the modification of that CI.

- Modify the "create" and "replace description" functions to allow the specification of a description at command entry time as an option to the current requirement of specifying the name of a file in which the description resides.

- Provide a project level option that modification request numbers are required for update of a CI only if that CI is not part of the current baseline. This would permit CIs in the initial development stage to be modified freely without having to specify an associated modification request number each time the CI is updated.

- Develop a mechanism for keeping track of interrelationships and dependencies among CIs. This extension may require the use of a DBMS and could be a rather substantial effort.

In addition to the above, the operational scope of the CI and Baseline Identification System can be modified to be something other than a UNIX file system by changing the setting for "home" in "main" to something other than the current file system. The program can be modified so that its scope is system wide by putting the "scm" directory under "/usr".

References

1. Begley, A. "The GTD-5 EAX Software Development Environment." GTE Automatic World-Wide Communication Journal, 19(November-December 1981):193-198

2. Bersoff, E. H. "Elements of Software Configuration Management." IEEE Transactions on Software Engineering, SE-10(January 1984):79-87

3. Bersoff, E. H., Henderson, V. D., and Siegel, S. G. Software Configuration Management An Investment in Product Integrity Englewood Cliffs, N.J.: Prentice-Hall, 1980

4. Bersoff, E. H., Henderson, V. D., and Siegel, S. G. "Software Configuration Management: A Tutorial." in Tutorial: Software Configuration Management." eds., W. Bryan, C. Chadbourne, and S. Siegel, Falls Church, Va.: Computer Society Press, 1980, pp.24-32

5. Boehm, B. W. "Verifying and Validating Software Requirements and Design Specifications." IEEE Software, 1(January 1984):75-88

6. Bryan, W. L., Siegel, S. G., and Whiteleather, G. L. "Auditing Throughout the Software Life Cycle: A Prime." Computer 15(March 1982):57-67

7. Bryan, W., Chadbourne, C., and Siegel, S., eds. Tutorial: Software Configuration Management, Falls Church, Va.: Computer Society Press, 1980, p.23

8. Dean, W. A. "Why Worry About Configuration Management?" Tutorial: Software Configuration Management, eds. W. Bryan, C. Chadbourne, and S. Siegel, Falls Church, Va.: Computer Society Press, 1980, pp.48-56

9. Knudsen, D. B., Barofsky, A. and Satz, L. R. "A Modification Request Control System." Proceedings of the 2nd International Conference on Software Engineering, 1976, pp.187-192

10. Martin, E. W. "The Context of STARS." Computer, 16(November 1983):14+

11. McCarthy, Rita "Applying the Technique of Configuration Management to Software." in Tutorial: Software Configuration Management, eds., W. Bryan, C. Chadbourne, and S. Siegel, Falls Church, Va.: Computer Society Press, 1980, pp.42-47

12. McCorduck, P. "Introduction to the Fifth Generation." Communications of the ACM, 26(September 1983):629-630

13. Prager, J. M. "The Project Automated Librarian." IBM Systems Journal, 22(1983):214-228

14. Rochkind, M. J. "The Source Code Control System." IEEE Transactions on Software Engineering, (December 1975):233-239

15. Tichy, W. F. "Design, Implementation, and Evaluation of a Revision Control System." Proceedings of the 6th International Conference on Software Engineering, 1982, pp.58-67

16. Support Tools Guide UNIX System. n.p.: Western Electric, 1982

17. UNIX System User's Manual Release 5.0. n.p.: Western Electric, 1982

Appendix A: Administrative Module Specifications

Module: supervisor

Inputs:

- a counter of the number of arguments passed to the scmid program
- a list of pointers to the passed arguments

Functions:

- determine the home path for the requester via the getenv subroutine.
- determine the SCM function requested by comparing the first argument with a list of valid SCM functions
- call the appropriate routine if a match is found
- return to caller if a match is not found

Outputs:

- a list of pointers to the arguments which are passed to the desired SCM function
- an error message to stderr if an unknown function was requested
- Return codes:
  - 0 => success
  - 16 => an unknown function was requested

Module: install

Inputs:

- a list of arguments resulting from a caller's "install [projid]" request:
  - project ID (optional)

Functions:

- create a SCM directory under the installer's login directory
- change the mode of the SCM directory to 770 so that the installer and other logins in the same group will have access to the directory
- create a project ID directory if one was specified in the argument list
- change the mode of the project ID directory to 770

Outputs:

- a SCM directory under the requester's login directory (see Figure 3-6)
- a project ID directory under the SCM directory if one was specified in the argument list (see Figure 3-6)
- error messages to stderr from mkdir and chmod if errors are encountered
- Return codes:
  - 0 => success
  - 2 => a project ID directory was not created
  - x => "x" return code from mkdir or chmod

Module: addproj

Inputs:

- a list of arguments resulting from a caller's "addproj projid" request:
  - the name of the project ID directory to be created

Functions:

- insure that 2 arguments were provided
- create a project ID directory
- change the mode of the project ID directory to 770

Outputs:

- error message to stderr if the argument count is less than 2
- a project ID directory under the requester's SCM directory (see Figure 3-7)
- error message to stderr from mkdir or chmod if errors are encountered
- Return codes:
  - 0 => success
  - 4 => insufficient arguments
  - x => "x" return code from mkdir or chmod

Module: addfile

Inputs:

- a list of arguments resulting from a caller's "addproj projid/projfile" request:
  - project ID
  - the name of the project file to be created

Functions:

- insure that 2 arguments were provided
- create a project file directory under the specified project ID directory
- change the mode of the project file directory to 700 to make the requester (administrator) the sole controller of the file (authorized users of the file will have access to it through the CI functions which are specified later).
- create a baseline directory under the project file directory

Outputs:

- error messages to stderr from mkdir and chmod if errors are encountered
- a project file directory under the specified project ID directory (see Figure 3-8)
- a baseline directory under the project file directory (see Figure 3-8)
- Return codes:
  - 0 => success
  - 4 => insufficient arguments
  - x => "x" return code from mkdir and chmod

Module: updatusr
  Inputs:
  • a list of arguments resulting from a caller's
    "updatusr projid/projfile ciname {-axxx -ayyy | -eaaa -eyyy}"
    request:
      - project ID
      - project file
      - CI name whose list of authorized users is to be modified
      - "-a" flag and login IDs of those users who are to be added to
        the list of authorized users for the specified CI
      - "-e" flag and login IDs of those users who are to be removed
        from the list of authorized users for the specified CI
  Functions:
  • insure that at least 3 arguments were provided
  • setup an argument list for SCCS admin
  • create a child process via fork
  • execute admin from the child process
  • wait for the child process to end
  • check admin status
  Outputs:
  • error message if fork, execv, or admin fail
  • an updated authorized user list in the SCCS file for the specified
    CI
  • Return codes:
      0 => success
      4 => insufficient arguments
     16 => fork failure
      x => 'x' return code from admin

Appendix B: Configuration Item Module Specifications

Module: createci

Inputs:

- a list of arguments resulting from a caller's "createci projid/projfile ciname descfile [-t??] [-v??]" request:
  - project ID
  - project file
  - name to given to the new CI
  - the file in which the CI's description can be found
  - optional CI type flag with type name
  - optional modification request validation flag with validation module name

Functions:

- insure that 3 arguments were provided
- check for valid flag specifications
- create the admin argument list
- create a child process via fork
- execute admin in the child via execv
- wait for admin to finish
- check status of admin

Outputs:

- error message to stderr if less than 3 input arguments
- an error message to stderr if an unknown flag is encountered
- error messages if fork, execv, or admin fail
- a SCCS file for the subject CI under the specified project file for the specified project ID
- Return codes:
  - 0 => success
  - 4 => insufficient arguments
  - 8 => argument error(s) detected
  - 16 => fork failure
  - x => 'x' return code from admin

Module: fetchci

Inputs:

- a list of arguments resulting from a user's "fetchci projid/projfile ciname [-e] [-rsid]" request:
  - project ID
  - project file
  - name of the CI to be retrieved
  - optional "retrieve for update" flag
  - optional "CI version to be fetched" flag plus desired SID
- the SCCS file for the desired CI

Functions:

- insure that a least 2 arguments are provided
- insure that each flag argument is correctly specified
- create the get argument list
- create a child process via fork
- execute get in the child via execv
- wait for get to finish
- check status of get

Outputs:

- error message to stderr if less than 2 input arguments
- error messages to stderr if flag specification errors are detected
- error message if fork, execv, or admin fail
- a copy of the requested CI in the requester's current directory
- Return codes:
  - 0 => success
  - 4 => insufficient arguments
  - 8 => argument error(s) detected
  - 16 => fork failure
  - x => 'x' returned from get

Module: updateci

Inputs:

- a list of arguments resulting from a user's "updateci projid/projfile ciname [-ycomments] [-m[mrlist]]" request:
  - project ID
  - project file
  - name of the CI to be updated
  - optional comment flag and comments
  - optional "modification request number" flag and optional list of mr numbers
- the copy of the CI that resides in the requester's current directory which will be used to created the new version of the CI

Functions:

- insure that a least 2 arguments are provided
- insure that each flag argument is correctly specified
- create the delta argument list
- create a child process via fork
- execute delta in the child via execv
- wait for delta to finish
- check status of delta

Outputs:

- error message to stderr if less than 2 input arguments
- error messages to stderr if flag argument specification errors are detected
- error message to stderr if fork, execv, or delta fail
- updated SCCS file for the CI
- Return codes:
  - 0 => success
  - 4 => insufficient arguments
  - 8 => argument error(s) detected
  - 16 => fork failure
  - x => 'x' returned from delta

Module: listci

Inputs:

- a list of arguments resulting from a user's "listci projid/projfile ciname [-c] [-d]" request:
  - project ID
  - project file
  - name of the CI to be listed
  - optional "list CI change history" flag
  - optional "list CI description" flag
- the SCCS file for the specified CI

Functions:

- insure that a least 2 arguments are provided
- insure that each flag argument is correctly specified
- set both flags if neither was specified in the argument list (i.e., produce both a description of the specified CI and its change history)
- create the prs argument list
- create a child process via fork
- execute prs in the child via execv
- wait for prs to finish
- check status of prs

Outputs:

- error message to stderr if less than 2 input arguments
- error messages to stderr if flag specification errors are detected
- error message to stderr if fork, execv, or prs fail
- the output requested to stdout
- Return codes:
  - 0 => success
  - 4 => insufficient arguments
  - 8 => argument error(s) detected
  - 16 => fork failure
  - x => 'x' returned from prs

Module: rcidesc
  Inputs:
  • a list of arguments resulting from a user's
    "rcidesc projid/projfile ciname descfile" request:
      - project ID
      - project file
      - name of the CI whose description is to be replaced
      - file name in which the new description can be found (this
        file must reside in the current directory of the requester or
        a complete path name must be specified)
  • the SCCS file for the specified CI
  Functions:
  • insure that a least 3 arguments are provided
  • create the admin argument list
  • create a child process via fork
  • execute admin in the child via execv
  • wait for admin to finish
  • check status of admin
  Outputs:
  • error message to stderr if less than 3 input arguments
  • error message to stderr if fork, execv, or admin fail
  • new description in the SCCS file for the specified CI
  • Return codes:
      0 => success
      4 => insufficient arguments
     16 => fork failure
      x => 'x' returned from admin

Module: civdesc
  Inputs:
  • a list of arguments resulting from a user's
    "civdesc projid/projfile ciname [-rsid] [-ycomments]" request:
      - project ID
      - project file
      - name of the CI whose specified SID description is to be amended
      - version flag and SID whose description is to be amended
      - optional comment flag and comments
  • the SCCS file for the specified CI
  Functions:
  • insure that a least 3 arguments are provided
  • insure that each flag argument is correctly specified
  • create the cdc argument list
  • create a child process via fork
  • execute cdc in the child via execv
  • wait for cdc to finish
  • check status of cdc
  Outputs:
  • error message to stderr if less than 3 input arguments
  • error message to stderr if flag specification errors are detected
  • error message to stderr if fork, execv, or cdc fail
  • amended SID description in the SCCS file of the specified CI
  • Return codes:
      0 => success
      4 => insufficient arguments
      8 => argument error(s) detected
     16 => fork failure
      x => 'x' returned from cdc

Appendix C: Baseline Module Specifications

Module: createbl

Inputs:

- a list of arguments resulting from a users "createbl projid/projfile blname descfile inexfile" request:
  - project ID
  - project file
  - name to be given to the baseline
  - the file name in which the baseline description can be found
  - a "CI by version" include/exclude file (format: flag {i|e} CIname release)
- all "s." files in the specified project ID/project file directory

Functions:

- insure that a least 3 arguments are provided
- get list of CIs and latest releases via prs
- sort and merge the list of CIs at the latest release with the list of CIs and versions to be include and/or exclude (inexfile)
- create the list of baseline CIs from the sorted and merged file
- create the admin argument list
- create a child process via fork
- execute admin in the child via execv
- wait for admin to finish
- check status of admin

Outputs:

- error message to stderr if less than 3 input arguments
- error message to stderr if prs or sort fail
- error message to stderr if fork, execv, or admin fail
- a SCCS file for the specified baseline in the baseline directory for the project file
- Return codes:
  - 0 => success
  - 4 => insufficient arguments
  - 16 => sort, prs, or fork problem
  - x => 'x' returned from admin

Module: fetchbl

Inputs:

- a list of arguments resulting from a user's "fetchbl projid/projfile blname [-e] [-rsid]" request:
  - project ID
  - project file
  - name of the baseline to be retrieved
  - optional "retrieve for update" flag
  - optional "baseline version to be fetched" flag plus desired SID
- the SCCS file for the desired baseline

Functions:

- insure that a least 2 arguments are provided
- insure that each flag argument is correctly specified
- create the get argument list
- create a child process via fork
- execute get in the child via execv
- wait for get to finish
- check status of get

Outputs:

- error message to stderr if less than 2 input arguments
- error messages to stderr if flag specification errors are detected
- error message if fork, execv, or admin fail
- a copy of the requested baseline in the requester's current directory
- Return codes:
  - 0 => success
  - 4 => insufficient arguments
  - 8 => argument error(s) detected
  - 16 => fork failure
  - x => 'x' returned from get

Module: updatebl

Inputs:

- a list of arguments resulting from a user's "updatebl projid/projfile blname [-ycomments]" request:
  - project ID
  - project file
  - name of the baseline to be updated
  - optional comment flag and comments
- the copy of the baseline that resides in the administrator's current directory which will be used to created the new version of the baseline

Functions:

- insure that a least 2 arguments are provided
- insure that each flag argument is correctly specified
- create the delta argument list
- create a child process via fork
- execute delta in the child via execv
- wait for delta to finish
- check status of delta

Outputs:

- error message to stderr if less than 2 input arguments
- error messages to stderr if flag argument specification errors are detected
- error message to stderr if fork, execv, or delta fail
- updated SCCS file for the baseline
- Return codes:
  - 0 => success
  - 4 => insufficient arguments
  - 8 => argument error(s) detected
  - 16 => fork failure
  - x => 'x' returned from delta

Module: listbl

Inputs:

- a list of arguments resulting from a user's "listbl projid/projfile blname [-c] [-d]" request:
  - project *ID*
  - project file
  - name of the baseline to be listed
  - optional "list baseline change history" flag
  - optional "list baseline description" flag
- the SCCS file for the specified baseline

Functions:

- insure that a least 2 arguments are provided
- insure that each flag argument is correctly specified
- set both flags if neither was specified in the argument list (i.e., produce both a description of the specified baseline and its change history)
- create the prs argument list
- create a child process via fork
- execute prs in the child via execv
- wait for prs to finish
- check status of prs

Outputs:

- error message to stderr if less than 2 input arguments
- error messages to stderr if flag specification errors are detected
- error message to stderr if fork, execv, or prs fail
- the output requested to stdout
- Return codes:
  - 0 => success
  - 4 => insufficient arguments
  - 8 => argument error(s) detected
  - 16 => fork failure
  - x => 'x' returned from prs

Module: rbldesc

Inputs:

- a list of arguments resulting from a user's "rbldesc projid/projfile blname descfile" request:
  - project ID
  - project file
  - name of the baseline whose description is to be replaced
  - file name in which the new description can be found (this file must reside in the current directory of the administrator or a complete path name must be specified)
- the SCCS file for the specified baseline

Functions:

- insure that a least 3 arguments are provided
- create the admin argument list
- create a child process via fork
- execute admin in the child via execv
- wait for admin to finish
- check status of admin

Outputs:

- error message to stderr if less than 3 input arguments
- error message to stderr if fork, execv, or admin fail
- new description in the SCCS file for the specified baseline
- Return codes:
  - 0 => success
  - 4 => insufficient arguments
  - 16 => fork failure
  - x => 'x' returned from admin

Module: blvdesc

Inputs:

- a list of arguments resulting from a user's "blvdesc projid/projfile blname [-rsid] [-ycomments]" request:
  - project ID
  - project file
  - name of the baseline whose specified SID description is to be amended
  - version flag and SID whose description is to be amended
  - optional comment flag and comments
- the SCCS file for the specified baseline

Functions:

- insure that a least 3 arguments are provided
- insure that each flag argument is correctly specified
- create the cdc argument list
- create a child process via fork
- execute cdc in the child via execv
- wait for cdc to finish
- check status of cdc

Outputs:

- error message to stderr if less than 3 input arguments
- error message to stderr if flag specification errors are detected
- error message to stderr if fork, execv, or cdc fail
- amended SID description in the SCCS file of the specified baseline
- Return codes:
  - 0 => success
  - 4 => insufficient arguments
  - 8 => argument error(s) detected
  - 16 => fork failure
  - x => 'x' returned from cdc

Appendix D: User's Guide

From a user's perspective, the CI and Baseline Identification System consists of 16 commands or functions. The functions are divided into *administrative, baseline, and CI functions.* Use of the administrative functions and most of the baseline functions are restricted to the designated SCM administrator for each file system. The administrative functions consist of the following:

```
install
addproj
addfile
updatusr
```

The baseline functions are:

```
createbl
fetchbl
updatebl
listbl
rbldesc
blvdesc
```

The CI commands can be used by all login names in a UNIX file system who are in the same group as the administrator. CI functions consist of the following:

```
createci
fetchci
updateci
listci
rcidesc
civdesc
```

A UNIX manual page has been prepared for each of the SCM identification functions. The remainder of this appendix consists of SCM manual pages in alphabetical order. It should be noted that all non-dash (i.e., non "-") parameters are positional in each of the SCM functions.

Name

addfile - create a project file directory under the specified project ID directory.

SYNOPSIS

addfile projid/projfile

DESCRIPTION

addfile will create a project file for projfile under the projid directory. Many project files can be created under each project ID directory.

Use of this command is restricted to the designated SCM administrator for a file system.

SEE ALSO

install(1), addproj(1), updatusr(1).

Name

addproj - create a project ID directory under the SCM directory of the caller's file system.

SYNOPSIS

addproj projid

DESCRIPTION

addproj will create a project ID directory for projfile under the SCM directory of the file system in which the command is issued. Many project ID directories can be created under each SCM directory.

Use of this command is restricted to the designated SCM administrator for a file system.

SEE ALSO

install(1), addfile(1), updatusr(1).

Name

blvdesc - amend the description of the specified version of a baseline.

SYNOPSIS

blvdesc projid/projfile blname -rsid [-ycomments]

DESCRIPTION

blvdesc will amend the description of the version of the blname specified by the -r parameter. Comments may be provided on the command line via the -y parameter. The caller will be prompted for comments if the -y parameter is not specified. If -y is specified, it must be the last parameter on the command line.

Use of this command is restricted to the SCM administrator.

SEE ALSO

createbl(1), fetchbl(1), updatebl(1), listbl(1), rbldesc(1).

Name

civdesc - amend the description of the specified version of a configuration item.

SYNOPSIS

civdesc projid/projfile ciname -rsid [-ycomments]

DESCRIPTION

civdesc will amend the description of the version of the ciname specified by the -r parameter. Comments may be provided on the command line via the -y parameter. The caller will be prompted for comments if the -y parameter is not specified. If -y is specified, it must be the last parameter on the command line.

SEE ALSO

createci(1), fetchci(1), updateci(1), listci(1), rcidesc(1).

Name

createbl - create a baseline in a SCM project file.

SYNOPSIS

createbl projid/projfile blname descfile [iefile]

DESCRIPTION

createbl will create a baseline module for the SCM projfile under SCM project projid. The description of the baseline will be taken from file descfile which must be in the current directory of the caller or be designated by its full path name. An optional include/exclude file, iefile, can be provided to override the normal creation process for a baseline. Entries in the iefile of the form "i ciname SID" will include the specified version of the designated CI instead of the most recent version (SID) for that CI. Entries in iefile of the form "e ciname" will exclude that CI from the baseline entirely.

Use of this command is restricted to the designated SCM administrator for a file system.

SEE ALSO

fetchbl(1), updatebl(1), listbl(1), rbldesc(1), blvdesc(1).

Name

createci - create a configuration item (CI) in a SCM project file.

SYNOPSIS

createci projid/projfile ciname descfile [-tType] [-v[module]]

DESCRIPTION

createci will create a CI module in the SCM projfile under SCM project projid. The description of the CI will be taken from file descfile which must be in the current directory of the caller or be designated by its full path name. An optional CI type can be specified via -tType. The caller may specify that modification request (MR) numbers are required to update this CI by specifying the -v option. The validation module (if one is specified) must reside in the same directory as the system's SCCS commands.

SEE ALSO

fetchci(1), updateci(1), listci(1), rcidesc(1), civdesc(1).

Name

fetchbl - retrieve a release of a baseline for information purposes or for subsequent update.

SYNOPSIS

fetchbl projid/projfile blname [-e] [-rsid]

DESCRIPTION

fetchbl will retrieve the latest version of blname from projfile which is under the projid directory and place it in the caller's current working directory. The -e parameter should be used if the subject baseline is to be subsequently updated. Otherwise the caller will receive a read-only version of the baseline. The caller may request a specific version of the baseline by specifying the -r parameter followed by the desired version number (SCCS SID).

Use of this command to retrieve a baseline for subsequent update is restricted to the SCM administrator. Other SCM users may retrieve baselines for information purposes.

SEE ALSO

createbl(1), updatebl(1), listbl(1), rbldesc(1), blvdesc(1).

Name

fetchci - retrieve a release of a configuration item for information purposes or for subsequent update.

SYNOPSIS

fetchci projid/projfile ciname [-e] [-rsid]

DESCRIPTION

fetchci will retrieve the latest version of ciname from projfile which is under the projid directory and place it in the caller's current working directory. The -e parameter should be used if the subject CI is to be subsequently updated. Otherwise the caller will receive a read-only version of the CI. The caller may request a specific version of the CI by specifying the -r parameter followed by the desired version number (SCCS SID).

SEE ALSO

createci(1), updateci(1), listci(1), rcidesc(1), civdesc(1).

Name

install - install the Configuration Item and Baseline Identification System under a UNIX file system directory.

SYNOPSIS

install [projid]

DESCRIPTION

install will install the SCM identification system under the UNIX file system in which the command is issued. A "scm" directory will be created under the file system directory and a SCM owner file "adm" will be created under the "scm" directory. The login name which issues this command will become the SCM administrator's login name for that file system. Therefore, use of this function will identify the SCM administrator for a file system as well as install the SCM system in a file system. A project ID directory for projid will be created under the SCM directory if specified.

The "adm" file will be used to restrict access to administrative and baseline functions to the SCM administrator once the system has been installed.

SEE ALSO

addproj(1), addfile(1), updatusr(1).

Name

listbl - list the description and/or change history for a baseline.

SYNOPSIS

listbl projid/projfile blname [-c] [-d]

DESCRIPTION

listbl will list the description and/or change history of blname which resides in projfile under the projid directory. The -c parameter will cause the change history to be listed. The -d parameter will cause the description of the baseline to be listed. If neither is specified, both the description and change history for the baseline will be listed. If a "." is specified for blname ,then all baselines in projfile will be listed.

All SCM users in a file system can use this command.

SEE ALSO

createbl(1), fetchbl(1), updatebl(1), rbldesc(1), blvdesc(1).

Name

listci - list the description and/or change history for a configuration item.

SYNOPSIS

listci projid/projfile ciname [-c] [-d]

DESCRIPTION

listci will list the description and/or change history of ciname which resides in projfile under the projid directory. The -c parameter will cause the change history to be listed. The -d parameter will cause the description of the CI to be listed. If neither is specified, both the description and change history for the CI will be listed. If a "." is specified for ciname ,then all CIs in projfile will be listed.

SEE ALSO

createci(1), fetchci(1), updateci(1), rcidesc(1), civdesc(1).

Name

rbldesc - replace the description of a specified baseline.

SYNOPSIS

rbldesc projid/projfile blname descfile

DESCRIPTION

rbldesc will replace the description of blname which resides in projfile under the projid directory with the description which is contained in file descfile.

Use of this command is restricted to the SCM administrator.

SEE ALSO

createbl(1), fetchbl(1), updatebl(1), listbl(1), blvdesc(1).

Name

rcidesc - replace the description of a specified configuration item.

SYNOPSIS

rcidesc projid/projfile ciname descfile

DESCRIPTION

rcidesc will replace the description of ciname which resides in projfile under the projid directory with the description which is contained in file descfile.

SEE ALSO

createci(1), fetchci(1), updateci(1), listci(1), civdesc(1).

Name

updatebl - update a baseline which has been previously retrieved for update (see fetchbl).

SYNOPSIS

updatebl projid/projfile blname [-ycomments]

DESCRIPTION

updatebl will update blname in projfile which resides under the projid directory. The subject baseline must have been previously "fetched for update" by the SCM administrator. The administrator will be prompted for comments unless the -y parameter is specified. If provided, the -y parameter must be the last parameter specified in the command line.

Use of this command is restricted to the SCM administrator.

SEE ALSO

createbl(1), fetchbl(1), listbl(1), rbldesc(1), blvdesc(1).

Name

updateci - update a configuration item which has been previously retrieved for update (see fetchci).

SYNOPSIS

updateci projid/projfile ciname [-m[mrnumber]] [-ycomments]

DESCRIPTION

updateci will update ciname in projfile which resides under the projid directory. The subject CI must have been previously "fetched for update" under the login name which issues the updateci command. If the -v parameter was specified at the time the CI was created (see createci), the -m parameter with or without a modification request number may be specified on the updateci command. The user will be prompted for MR numbers if the -m parameter is not specified and -v was specified when the CI was created via createci. The caller will also be prompted for comments unless the -y parameter is specified. If provided, the -y parameter must be the last parameter specified in the command line.

SEE ALSO

createci(1), fetchci(1), listci(1), rcidesc(1), civdesc(1).

Name

updatusr - update the authorized user list for a specified CI.

SYNOPSIS

updatusr projid/projfile ciname {-axxx | -eyyy}

DESCRIPTION

updatusr will change the list of authorized users for a specified CI. A login name can be added as an authorized user by specifying the -a parameter followed by the login name. Similarly, a login name can be removed from the authorized user list of a CI by specifying -e followed by the login name. More than one login name can be added and/or removed from an authorized user list at a time. Each such request requires a login name preceded by the appropriate dash parameter.

Use of this command is restricted to the designated SCM administrator for each file system.

SEE ALSO

install(1), addproj(1), addfile(1).

Appendix E: Source Code

```
#include <stdio.h>
#define INTBYTES 4    /* number of bytes per integer */
#define MAXPATH  70   /* max path length */
#define MAXCMD   80   /* max command length */
#define MAXARGS 30    /* max number of SCCS argruments */
#define MAXLINE 80    /* max number of characters in a line */

/* Define an external name table of legitmated SCM commands.
   This table is an array of pointers to the character array
   which follows.                                              */
char *namet[] =
     {"install",
      "addproj",
      "addfile",
      "updatusr",
      "createci",
      "fetchci",
      "updateci",
      "listci",
      "rcidesc",
      "civdesc",
      "createbl",
      "fetchbl",
      "updatebl",
      "listbl",
      "rbldesc",
      "blvdesc",
      "invalid"};

/* Define all functions as integer functions
   This definition is required in order to define an external branch table.*/
int install(),
    addproj(),
    addfile(),
    updatusr(),
    createci(),
    fetchci(),
    updateci(),
    listci(),
    rcidesc(),
    civdesc(),
    createbl(),
    fetchbl(),
    updatebl(),
    listbl(),
```

```
    rbldesc(),
    blvdesc(),
    invalid();

/* Define an external branch table for the functional routines
   This table is an array of pointers to the integer functions specified
   in the array and defined above.                                    */
int (*brancht[]) () =
    {install,
     addproj,
     addfile,
     updatusr,
     createci,
     fetchci,
     updateci,
     listci,
     rcidesc,
     civdesc,
     createbl,
     fetchbl,
     updatebl,
     listbl,
     rbldesc,
     blvdesc,
     invalid};

int *getenv();  /* getenv is a function which returns a pointer of int */
char scm[] = "/scm";
char mkdir[] = "mkdir  ";
char cm770[] = "chmod 770 ";
char cm700[] = "chmod 700 ";
char bldir[] = "bldir";
char sprefix[] = "s.";
char star[] = "*";
char slash[] = "/";
char blank[] = " ";
char dquote[] = "\042";
char nline[] = "\n";
char home[100];
char filesys[20];
char logname[50];
char owner[50];
int maxfunc, thispid;

char *xargv[MAXARGS];
int xargc;

/* command names */
char admin[] = "admin";
char get[] = "get";
```

```
char delta[] = "delta";
char cdc[] = "cdc";
char prs[] = "prs";

/* prs specifications */
char cprs[] = "\nVersion: :Dt:\nComments: :C:\nMR's: :MR:\n";
char cprs2[] = "\nModule: :M: Version: :Dt:\nComments: :C:\nMR's: :MR:\n";
char dprs[] = "\nModule: :M:  Type: :Y:\nDescription:\n  :FD:\nList of autho
rized users:\n  :UN:\n";
char blcprs[] = "\nVersion: :Dt:\nComments: :C:";
char blcprs2[] = "\nModule: :M: Version: :Dt:\nComments: :C:";
char bldprs[] = "\nModule: :M:\n  Description:\n  :FD:\nList of authorized u
sers:\n  :UN:\n";

/* shell program specicifications */
char blsort[] = "sort +1 +0 -1 -o t.srtblist t.blist ";
char blprs[] = "prs -d\"z :M: :I:\" -r ";
char blist[]  = " >t.blist";

main(argc,argv)
int argc;
char *argv[];
{

/* Define other needed variable */
FILE *admfp;
char line[80], adm[80];
int i, scnt, offset, rc, strcmp();
int *ptr;

/* Supervisor */
maxfunc = (sizeof(brancht)/INTBYTES);
ptr = getenv("HOME");
strcpy(home,ptr);
ptr = getenv("LOGNAME");
strcpy(logname,ptr);


/* determine file system name */
scnt = i = 0;
while ((scnt < 2) && (home[i] != '\0'))
     {
      if (home[i] == '\/')
         scnt++;
      filesys[i] = home[i];
     i++;
     }
filesys[--i] = '\0';

/* read adm file to determine the administrator name */
```

```
if (strcmp(argv[0],"install") != 0)
   {
    strcpy(adm,filesys);
    strcat(adm,scm);
    strcat(adm,slash);
    strcat(adm,"adm");
    if ((admfp = fopen(adm,"r")) == NULL)
       {fprintf(stderr,"\n**********  ERROR  **********\n");
        fprintf(stderr," can not open /filesys/scm/adm.\n");
        fprintf(stderr,"**********************************\n");
        return(16);
       }
    if (fgets(line,79,admfp) == NULL)
       {fprintf(stderr,"\n**********  ERROR  **********\n");
        fprintf(stderr," /filesys/scm/adm is empty.\n");
        fprintf(stderr,"**********************************\n");
        return(16);
       }
    sscanf(line,"%s",owner);
    fclose(admfp);
   }

/* reset home to be the file system directory */
strcpy(home,filesys);

/* Determine the appropriate function to call */
offset = maxfunc;
for (i=0;i<maxfunc;i++)
    {
      if(strcmp(argv[0],namet[i]) == 0)
       {
         offset = i;
         i = maxfunc;
       }
    }
if (offset == maxfunc)
   {
   /* Bad function name received */
   fprintf(stderr,"\n**********  ERORR  **********\n");
   fprintf(stderr,"Invalid function name encountered. \n");
   fprintf(stderr,"Probably an installation error. \n");
   fprintf(stderr,"Review your installation procedure. \n");
   fprintf(stderr,"**********************************\n");
   exit(1);
   }

/* Call the appropriate function */
rc = (*brancht[offset]) (--argc,&argv[1]);
exit(rc);
}
```

```
/*   END OF MAIN   */

/* "invalid" is a dummy function.  It is used as a means to
   determine when scmid has been entered with a bad function
   name.                                                    */
invalid () {}

cmderr(cmd)
char cmd[];
{fprintf(stderr,"\n************  ERROR  ************\n");
 fprintf(stderr,"The following command failed:\n");
 fprintf(stderr,"   %s\n",cmd);
 fprintf(stderr,"**********************************\n");
}

aout()
{
    {
    }
}
/*   START OF INSTALL   */
install(argc,argv)
int argc;
char *argv[];
{
FILE *admfp;
int i, rc;
char path[MAXPATH], cmd[MAXCMD], line[80], adm[80];
strcpy(path,home);

/* mkdir /HOME/scm */
strcat(path,scm);
strcpy(cmd,mkdir);
strcat(cmd,path);
rc = system(cmd);
if (rc != 0)
   {cmderr(cmd);
    return(rc);
   }

/* chmod 770 /HOME/scm */
strcpy(cmd,cm770);
strcat(cmd,path);
rc = system(cmd);
if (rc != 0)
   {cmderr(cmd);
    return(rc);
   }

/* create an adm file under the SCM directory */
```

```
strcpy(adm,path);          /* /HOME/scm */
strcat(adm,slash);
strcat(adm,"adm");         /* /HOME/scm/adm */
admfp = fopen(adm,"w");
if (admfp == NULL)
   {fprintf(stderr,"\n***********  ERROR  ***********\n");
    fprintf(stderr," can not open /filesys/scm/adm.\n");
    fprintf(stderr,"*********************************\n");
    return(16);
   }
sprintf(line,"%s",logname);
fputs(line,admfp);

/* chmod 700 /HOME/scm/adm */
strcpy(cmd,cm700);
strcat(cmd,adm);
rc = system(cmd);
if (rc != 0)
   {cmderr(cmd);
    return(rc);
   }

/* mkdir for project ID */
if (*argv[0] == NULL)
   return(2);
strcat(path,slash);
strcat(path,argv[0]);
strcpy(cmd,mkdir);
strcat(cmd,path);
rc = system(cmd);
if (rc != 0)
   {cmderr(cmd);
    return(rc);
   }
/* chmod 770 /HOME/scm/projectID */
strcpy(cmd,cm770);
strcat(cmd,path);
rc = system(cmd);
if (rc != 0)
   {cmderr(cmd);
    return(rc);
   }
return(0);
}
/*  END OF INSTALL   */

/*   START OF ADDPROJ   */
addproj(argc,argv)
int argc;
char *argv[];
```

```
{
int i, rc;
char path[MAXPATH], cmd[MAXCMD];

/* check authorization */
if (strcmp(owner,logname) != 0)
   {fprintf(stderr,"\n***********  ERROR  ***********\n");
    fprintf(stderr," This function is retricted for \n");
    fprintf(stderr," use to the administrator.\n");
    fprintf(stderr,"*********************************\n");
    return(256);
   }
if (*argv[0] == NULL)
   {fprintf(stderr,"***********  ERROR  ***********\n");
    fprintf(stderr,"Project ID required with addproj\n");
    fprintf(stderr,"*********************************\n");
    return(4);
   }

/* create project ID directory */
strcpy(path,home);
strcat(path,scm);
strcat(path,slash);
strcat(path,argv[0]);
strcpy(cmd,mkdir);
strcat(cmd,path);
rc = system(cmd);
if (rc != 0)
   {cmderr(cmd);
    return(rc);
   }

/* chmod 770 /HOME/scm/projid */
strcpy(cmd,cm770);
strcat(cmd,path);
rc = system(cmd);
if (rc != 0)
   {cmderr(cmd);
    return(rc);
   }
return(0);
}
/*  END OF ADDPROJ   */

addfile(argc,argv)
char *argv[];
{
int i, rc;
char path[MAXPATH], cmd[MAXCMD];
```

```
/* check authorization */
if (strcmp(owner,logname) != 0)
   {fprintf(stderr,"\n***********  ERROR  ***********\n");
    fprintf(stderr," This function is retricted for \n");
    fprintf(stderr," use to the administrator.\n");
    fprintf(stderr,"*********************************\n");
    return(256);
   }
if (*argv[0] == NULL)
   {fprintf(stderr,"\n***********  ERROR  ***********\n");
    fprintf(stderr,"projectid/filename required for addfile\n");
    fprintf(stderr,"*********************************\n");
    return(4);
   }

/* create file directory */
strcpy(path,home);
strcat(path,scm);
strcat(path,slash);
strcat(path,argv[0]);
strcpy(cmd,mkdir);
strcat(cmd,path);
rc = system(cmd);
if (rc != 0)
   {cmderr(cmd);
    return(rc);
   }

/* set mode to 700 */
strcpy(cmd,cm700);
strcat(cmd,path);
rc = system(cmd);
if (rc != 0)
   {cmderr(cmd);
    return(rc);
   }

/* create a baseline directory under /HOME/scm/projid/projfile */
strcat(path,slash);
strcat(path,bldir);
strcpy(cmd,mkdir);
strcat(cmd,path);
rc = system(cmd);
if (rc != 0)
   {cmderr(cmd);
    return(rc);
   }

/* set mode to 700 */
strcpy(cmd,cm700);
```

```
strcat(cmd,path);
rc = system(cmd);
if (rc != 0)
   {cmderr(cmd);
    return(rc);
   }
return(0);
}

updatusr(argc,argv)
char *argv[];
{
int i, rc, pid, wpid, stat;

char path[MAXPATH], cmd[MAXCMD];

/* check authorization */
if (strcmp(owner,logname) != 0)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," This function is retricted for \n");
    fprintf(stderr," use to the administrator.\n");
    fprintf(stderr,"**********************************\n");
    return(256);
   }
stat = 0;
if (argc < 3)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," updatusr requires at least 3 arguments.\n");
    fprintf(stderr,"**********************************\n");
    return(4);
   }

/*create expanded path */
strcpy(path,home);
strcat(path,scm);
strcat(path,slash);
strcat(path,argv[0]);
strcat(path,slash);
strcat(path,sprefix);
strcat(path,argv[1]);

/* set argv[0] to point to "admin" */
argv[0] = admin;

/* set argv[1] to point to path */
argv[1] = path;

/* fork to create a child process */
pid = fork();
if (pid == -1)
```

```
   {fprintf(stderr,"Can't service fork request at this time.\n");
    fprintf(stderr,"Try updatusr again later.\n");
    return(16);
   }

/* child process => execute SCCS command */
if (pid == 0)
   execv("/usr/bin/admin",argv);

/* wait in parent process for child to finish */
while ((wpid = wait(&stat)) != pid);
if (stat != 0)
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr,"SCCS admin request failed.\n");
    fprintf(stderr,"**********************************\n");
    return(stat);
   }
return(0);
}

createci(argc,argv)
int argc;
char *argv[];
{
int i, j, rc, pid, wpid, stat, cnt;

char path[MAXPATH], *sptr, a[10][50];
stat = 0;
cnt = 0;

/* check the number of arguments supplied */
if (argc < 3)
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr," createci requires at least 3 arguments.\n");
    fprintf(stderr,"**********************************\n");
    return(4);
   }

/* insure 3rd argument dose not start with "-" */
if (*argv[2] == '-')
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr," 3rd argument must be a filename.\n");
    fprintf(stderr,"**********************************\n");
    return(8);
   }

/* copy argument pointers to SCCS argrument pointer list */
for (i=0; i<argc ; i++)
    xargv[i] = argv[i];
xargc = argc;
```

```
/* check for valid flags in the argument list and expand '-' arguments */
i = xargc;
rc = j = 0;
while (--i > 0)
   if (*xargv[++j] == '-')
     {
      sptr = xargv[j]+1;
      switch(*sptr)
      {
       case 't':
         a[cnt][0] = '-';
         a[cnt][1] = 'f';
         strcpy(&a[cnt][2],sptr);
         xargv[j] = a[cnt];
         cnt++;
         break;
       case 'v':
         a[cnt][0] = '-';
         a[cnt][1] = 'f';
         strcpy(&a[cnt][2],sptr);
         xargv[j] = a[cnt];
         cnt++;
         a[cnt][0] = '-';
         a[cnt][1] = 'm';
         a[cnt][2] = '\0';
         xargv[xargc++] = a[cnt];
         cnt++;
         break;
       default:
         fprintf(stderr,"\n***********  ERROR  ************\n");
         fprintf(stderr," Invalid '-' option encountered\n");
         fprintf(stderr,"************************************\n");
         rc = 8;
         break;
      }
   }
if (rc != 0)
   return(rc);

/* set '-i' and '-n' arguments */
a[cnt][0] = '-';
a[cnt][1] = 'i';
strcpy(&a[cnt][2],argv[1]);
xargv[xargc++] = a[cnt];
cnt++;
a[cnt][0] = '-';
a[cnt][1] = 'n';
a[cnt][2] = '\0';
xargv[xargc++] = a[cnt];
cnt++;
```

```
/* create arguement list for admin */
strcpy(path,home);          /* HOME */
strcat(path,scm);          /* HOME/scm */
strcat(path,slash);
strcat(path,xargv[0]);          /* HOME/scm/projid/projfile */
strcat(path,slash);
strcat(path,sprefix);
strcat(path,xargv[1]);          /* HOME/scm/projid/projfile/s.ciname */

/* set xargv[0] to point to "admin" */
xargv[0] = admin;

/* set xargv[1] to point to path */
xargv[1] = path;

/* put '-t' prefix on descfile */
a[cnt][0] = '-';
a[cnt][1] = 't';
strcpy(&a[cnt][2],xargv[2]);
xargv[2] = a[cnt];
cnt++;

/* put creator's LOGNAME in the list of authorized users */
a[cnt][0] = '-';
a[cnt][1] = 'a';
strcpy(&a[cnt][2],logname);
xargv[xargc++] = a[cnt];
cnt++;


/* fork to create a child process */
pid = fork();
if (pid == -1)
   {fprintf(stderr,"Can't service fork request at this time.\n");
    fprintf(stderr,"Try createci again later.\n");
    return(16);
   }

/* child process => execute SCCS command */
if (pid == 0)
   execv("/usr/bin/admin",xargv);

/* wait in parent process for child to finish */
while ((wpid = wait(&stat)) != pid);
if (stat != 0)
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr,"SCCS admin request failed.\n");
    fprintf(stderr,"**********************************\n");
    return(stat);
   }
```

```
return(0);
}

fetchci(argc,argv)
int argc;
char *argv[];
{
int i, j, rc, pid, wpid, stat, cnt;

char path[MAXPATH], *sptr;
stat = 0;

/* check the number of arguments supplied */
if (argc < 2)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," fetchci requires at least 2 arguments.\n");
    fprintf(stderr,"**********************************\n");
    return(4);
   }

/* copy argument pointers to SCCS argrument pointer list */
for (i=0; i<argc ; i++)
    xargv[i] = argv[i];
xargc = argc;

/* check for valid flags in the argument */
i = xargc;
rc = j = 0;
while (--i > 0)
   if (*xargv[++j] == '-')
     {
      sptr = xargv[j]+1;
      switch(*sptr)
      {
       case 'e':
         break;
       case 'r':
         break;
       default:
         fprintf(stderr,"\n***********  ERROR  ************\n");
         fprintf(stderr," Invalid '-' option encountered\n");
         fprintf(stderr,"**********************************\n");
         rc = 8;
         break;
      }
   }
if (rc != 0)
   return(rc);

/* create arguement list for admin */
```

```
strcpy(path,home);          /* HOME */
strcat(path,scm);          /* HOME/scm */
strcat(path,slash);
strcat(path,xargv[0]);          /* HOME/scm/projid/projfile */
strcat(path,slash);
strcat(path,sprefix);
strcat(path,xargv[1]);          /* HOME/scm/projid/projfile/s.ciname */

/* set xargv[0] to point to "get" */
xargv[0] = get;

/* set xargv[1] to point to path */
xargv[1] = path;


/* fork to create a child process */
pid = fork();
if (pid == -1)
   {fprintf(stderr,"Can't service fork request at this time.\n");
    fprintf(stderr,"Try fetchci again later.\n");
    return(16);
   }

/* child process => execute SCCS command */
if (pid == 0)
   execv("/usr/bin/get",xargv);

/* wait in parent process for child to finish */
while ((wpid = wait(&stat)) != pid);
if (stat != 0)
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr,"SCCS get request failed.\n");
    fprintf(stderr,"**********************************\n");
    return(stat);
   }
return(0);
}

updateci(argc,argv)
int argc;
char *argv[];
{
int i, j, k, rc, pid, wpid, stat, cnt;

char path[MAXPATH], *sptr, cmnt[200];
stat = 0;

/* check the number of arguments supplied */
if (argc < 2)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
```

```
    fprintf(stderr," updateci requires at least 2 arguments.\n");
    fprintf(stderr,"**********************************\n");
    return(4);
   }

/* copy argument pointers to SCCS argrument pointer list */
for (i=0; i<argc ; i++)
    xargv[i] = argv[i];
xargc = argc;

/* check for valid flags in the argument */
i = xargc;
rc = j = 0;
while (--i > 0)
   if (*xargv[++j] == '-')
     {
      sptr = xargv[j]+1;
      switch(*sptr)
      {
       case 'y':
         k = j;
         strcpy(cmnt,xargv[j]);
         while (--i > 0)
           {
            strcat(cmnt,blank);
            strcat(cmnt,xargv[++k]);
            xargc--;
           }
         xargv[j] = cmnt;
         xargv[++j] = NULL;
         break;
       case 'm':
         break;
       default:
         fprintf(stderr,"\n***********  ERROR  ************\n");
         fprintf(stderr," Invalid '-' option encountered\n");
         fprintf(stderr,"**********************************\n");
         rc = 8;
         break;
      }
   }
if (rc != 0)
   return(rc);

/* create arguement list for admin */
strcpy(path,home);          /* HOME */
strcat(path,scm);          /* HOME/scm */
strcat(path,slash);
strcat(path,xargv[0]);          /* HOME/scm/projid/projfile */
strcat(path,slash);
```

```
strcat(path,sprefix);
strcat(path,xargv[1]);          /* HOME/scm/projid/projfile/s.ciname */

/* set xargv[0] to point to "delta" */
xargv[0] = delta;

/* set xargv[1] to point to path */
xargv[1] = path;


/* fork to create a child process */
pid = fork();
if (pid == -1)
   {fprintf(stderr,"Can't service fork request at this time.\n");
    fprintf(stderr,"Try updateci again later.\n");
    return(16);
   }

/* child process => execute SCCS command */
if (pid == 0)
   execv("/usr/bin/delta",xargv);

/* wait in parent process for child to finish */
while ((wpid = wait(&stat)) != pid);
if (stat != 0)
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr,"SCCS delta request failed.\n");
    fprintf(stderr,"**********************************\n");
    return(stat);
   }
return(0);
}

listci(argc,argv)
int argc;
char *argv[];
{
int i, j, rc, pid, wpid, stat, cnt, cflag, dflag;

char path[MAXPATH], *sptr, argstr[300], l[3], e[3], a[3];
stat = 0;
cflag = dflag = 0;

/* check the number of arguments supplied */
if (argc < 2)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," listci requires at least 2 arguments.\n");
    fprintf(stderr,"**********************************\n");
    return(4);
   }
```

```
/* copy argument pointers to SCCS argrument pointer list */
for (i=0; i<argc ; i++)
    xargv[i] = argv[i];
xargc = argc;

/* check for valid flags in the argument */
i = xargc;
rc = j = 0;
while (--i > 0)
   if (*xargv[++j] == '-')
     {
      sptr = xargv[j]+1;
      switch(*sptr)
      {
       case 'c':
         cflag = 1;
         break;
       case 'd':
         dflag = 1;
         break;
       default:
         fprintf(stderr,"\n***********  ERROR  ************\n");
         fprintf(stderr," Invalid '-' option encountered\n");
         fprintf(stderr,"***********************************\n");
         rc = 8;
         break;
      }
   }
if (rc != 0)
   return(rc);

/* create arguement list for admin */
strcpy(path,home);          /* HOME */
strcat(path,scm);          /* HOME/scm */
strcat(path,slash);
strcat(path,xargv[0]);          /* HOME/scm/projid/projfile */
if (*xargv[1] != '.')
   {
    strcat(path,slash);
    strcat(path,sprefix);
    strcat(path,xargv[1]);          /* HOME/scm/projid/projfile/s.ciname */
   }

/* set xargv[0] to point to "prs" */
xargv[0] = prs;

/* set xargv[1] to point to path */
xargv[1] = path;

/* determine output required */
```

```
argstr[0] = '-';
argstr[1] = 'd';
if (cflag == dflag)
    cflag = dflag = 1;
if (dflag == 1)
   {
    strcpy(&argstr[2],dprs);
    xargv[2] = argstr;
    xargv[3] = NULL;
    xargc = 3;
    /* fork to create a child process */
    pid = fork();
    if (pid == -1)
       {fprintf(stderr,"Can't service fork request at this time.\n");
        fprintf(stderr,"Try listci again later.\n");
        return(16);
       }
    /* child process => execute SCCS command */
    if (pid == 0)
       execv("/usr/bin/prs",xargv);

    /* wait in parent process for child to finish */
    while ((wpid = wait(&stat)) != pid);
    if (stat != 0)
       {fprintf(stderr,"\n************  ERROR  ************\n");
        fprintf(stderr,"SCCS prs request failed.\n");
        fprintf(stderr,"**********************************\n");
        return(stat);
       }
   }
if (cflag == 1)
   {
    l[0] = '-';
    l[1] = 'l';
    l[2] = '\0';
    e[0] = '-';
    e[1] = 'e';
    e[2] = '\0';
    a[0] = '-';
    a[1] = 'a';
    a[2] = '\0';
    if (dflag == 1)
       strcpy(&argstr[2],cprs);
    else strcpy(&argstr[2],cprs2);
    xargv[2] = argstr;
    xargc++;
    xargv[3] = l;
    xargc++;
    xargv[4] = e;
    xargc++;
```

```
    xargv[5] = a;
    xargv[6] = NULL;
    xargc = 6;
    /* fork to create a child process */
    pid = fork();
    if (pid == -1)
       {fprintf(stderr,"Can't service fork request at this time.\n");
        fprintf(stderr,"Try listci again later.\n");
        return(16);
       }
    /* child process => execute SCCS command */
    if (pid == 0)
       execv("/usr/bin/prs",xargv);
    /* wait in parent process for child to finish */
    while ((wpid = wait(&stat)) != pid);
    if (stat != 0)
       {fprintf(stderr,"\n************  ERROR  ************\n");
        fprintf(stderr,"SCCS prs request failed.\n");
        fprintf(stderr,"**********************************\n");
        return(stat);
       }
   }
return(0);
}

rcidesc(argc,argv)
int argc;
char *argv[];
{
int i, j, k, rc, pid, wpid, stat, cnt;

char path[MAXPATH], *sptr, a[100];
stat = 0;

/* check the number of arguments supplied */
if (argc < 3)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," rcidesc requires at least 3 arguments.\n");
    fprintf(stderr,"**********************************\n");
    return(4);
   }

/* copy argument pointers to SCCS argrument pointer list */
for (i=0; i<argc ; i++)
    xargv[i] = argv[i];
xargc = argc;

/* create arguement list for admin */
strcpy(path,home);          /* HOME */
strcat(path,scm);          /* HOME/scm */
```

```
strcat(path,slash);
strcat(path,xargv[0]);          /* HOME/scm/projid/projfile */
strcat(path,slash);
strcat(path,sprefix);
strcat(path,xargv[1]);          /* HOME/scm/projid/projfile/s.ciname */

/* set xargv[0] to point to "admin" */
xargv[0] = admin;

/* set xargv[1] to point to path */
xargv[1] = path;

/* put "-t" prefix on descfile */
a[0] = '-';
a[1] = 't';
strcpy(&a[2],xargv[2]);
xargv[2] = a;


/* fork to create a child process */
pid = fork();
if (pid == -1)
   {fprintf(stderr,"Can't service fork request at this time.\n");
    fprintf(stderr,"Try rcidesc again later.\n");
    return(16);
   }

/* child process => execute SCCS command */
if (pid == 0)
   execv("/usr/bin/admin",xargv);

/* wait in parent process for child to finish */
while ((wpid = wait(&stat)) != pid);
if (stat != 0)
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr,"SCCS admin request failed.\n");
    fprintf(stderr,"**********************************\n");
    return(stat);
   }
return(0);
}

civdesc(argc,argv)
int argc;
char *argv[];
{
int i, j, k, rc, pid, wpid, stat, cnt;

char path[MAXPATH], *sptr, cmnt[200];
stat = 0;
```

```
/* check the number of arguments supplied */
if (argc < 3)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," civdesc requires at least 3 arguments.\n");
    fprintf(stderr,"**********************************\n");
    return(4);
   }

/* copy argument pointers to SCCS argrument pointer list */
for (i=0; i<argc ; i++)
    xargv[i] = argv[i];
xargc = argc;

/* check for valid flags in the argument */
i = xargc;
rc = j = 0;
while (--i > 0)
   if (*xargv[++j] == '-')
     {
      sptr = xargv[j]+1;
      switch(*sptr)
      {
       case 'y':
         k = j;
         strcpy(cmnt,xargv[j]);
         while (--i > 0)
           {
            strcat(cmnt,blank);
            strcat(cmnt,xargv[++k]);
            xargc--;
           }
         xargv[j] = cmnt;
         xargv[++j] = NULL;
         break;
       case 'r':
         break;
       default:
         fprintf(stderr,"\n***********  ERROR  ************\n");
         fprintf(stderr," Invalid '-' option encountered\n");
         fprintf(stderr,"**********************************\n");
         rc = 8;
         break;
      }
   }
if (rc != 0)
   return(rc);

/* create arguement list for cdc */
strcpy(path,home);          /* HOME */
strcat(path,scm);          /* HOME/scm */
```

```
strcat(path,slash);
strcat(path,xargv[0]);          /* HOME/scm/projid/projfile */
strcat(path,slash);
strcat(path,sprefix);
strcat(path,xargv[1]);          /* HOME/scm/projid/projfile/s.ciname */

/* set xargv[0] to point to "cdc" */
xargv[0] = cdc;

/* set xargv[1] to point to path */
xargv[1] = path;


/* fork to create a child process */
pid = fork();
if (pid == -1)
   {fprintf(stderr,"Can't service fork request at this time.\n");
    fprintf(stderr,"Try civdesc again later.\n");
    return(16);
   }

/* child process => execute SCCS command */
if (pid == 0)
   execv("/usr/bin/cdc",xargv);

/* wait in parent process for child to finish */
while ((wpid = wait(&stat)) != pid);
if (stat != 0)
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr,"SCCS cdc request failed.\n");
    fprintf(stderr,"**********************************\n");
    return(stat);
   }
return(0);
}

createbl(argc,argv)
int argc;
char *argv[];
{
FILE *inptr, *outptr;
int i, rc, pid, wpid, stat, x, *ptr, cnt;

char path[MAXPATH], *sptr, cmd[MAXCMD], line[80], a[10][50];

struct cilist
   { char action;
     char ciname[30];
     char rel[8];
   };
```

```
struct cilist last, cur, *sp;


/* check authorization */
if (strcmp(owner,logname) != 0)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," This function is retricted for \n");
    fprintf(stderr," use to the administrator.\n");
    fprintf(stderr,"**********************************\n");
    return(256);
   }
cnt = 0;

/* check the number of arguments supplied */
if (argc < 3)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," createbl requires at least 3 arguments.\n");
    fprintf(stderr,"**********************************\n");
    return(4);
   }

/* copy argument pointers to SCCS argrument pointer list */
for (i=0; i<argc ; i++)
    xargv[i] = argv[i];
xargc = argc;

/* create path for prs execution */
strcpy(path,home);          /* HOME */
strcat(path,scm);          /* HOME/scm */
strcat(path,slash);
strcat(path,xargv[0]);          /* HOME/scm/projid/projfile */
strcat(path,slash);
strcat(path,sprefix);
strcat(path,star);               /* HOME/scm/projid/projfile/s.* */

/* create t.blist file via system */
strcpy(cmd,blprs);
strcat(cmd,path);
strcat(cmd,blist);
rc = system(cmd);
if (rc == -1)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," execution of prs failed.\n");
    fprintf(stderr,"**********************************\n");
    return(16);
   }

 /* sort and merge t.blist with inexfile (3rd arg) to create t.srtblist */
if (argc >= 4)
   {
```

```
    strcpy(cmd,blsort);
    strcat(cmd,xargv[3]);
    rc = system(cmd);
    if (rc == -1)
       {fprintf(stderr,"\n***********  ERROR  ************\n");
        fprintf(stderr," execution of sort failed.\n");
        fprintf(stderr,"************************************\n");
        return(16);
       }
    /* remove t.blist */
    rc = system("rm t.blist");
    if (rc == -1)
       {fprintf(stderr,"\n***********  WARNING  ************\n");
        fprintf(stderr," execution of 'rm t.blist' failed.\n");
        fprintf(stderr,"************************************\n");
       }

   }
else {
      rc = system("mv t.blist t.srtblist");
      if (rc == -1)
         {fprintf(stderr,"\n***********  WARNING  ************\n");
          fprintf(stderr," execution of 'mv t.blist t.srtblist' failed.\n");
          fprintf(stderr,"************************************\n");
         }
     }

/* create t.blinput from t.srtblist */
if ((inptr = fopen("t.srtblist","r")) == NULL)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," can not open t.srtblist.\n");
    fprintf(stderr,"************************************\n");
    return(16);
   }
else {
      outptr = fopen("t.blinput","w");
      if (outptr == NULL)
         {fprintf(stderr,"\n***********  ERROR  ************\n");
          fprintf(stderr," open failed for t.blinput.\n");
          fprintf(stderr,"************************************\n");
          return(16);
         }
     sp = &cur;
     while ((fgets(line,MAXLINE-1,inptr)) != NULL)
        {
         cur.action = '\0';
         strcpy(cur.ciname,"\0");
         strcpy(cur.rel,"\0");
         x=sscanf(line,"%c %s %s",&((*sp).action), (*sp).ciname, (*sp).rel);
         if ((cur.action == 'i') && (x == 3))
```

```
            {
             last.action = cur.action;
             strcpy(last.ciname,cur.ciname);
             strcpy(last.rel,cur.rel);
             sprintf(line,"%s %s %s",(*sp).ciname, (*sp).rel, nline);
             fputs(line,outptr);
             continue;
            }
         if ((cur.action == 'e') && (x == 2))
            {
             last.action = cur.action;
             strcpy(last.ciname,cur.ciname);
             strcpy(last.rel,cur.rel);
             continue;
            }
         if ((cur.action == 'z') && (x == 3))
            {
             rc = strcmp(cur.ciname,last.ciname);
             if (rc == 0)
                continue;
             else
               {
                  last.action = cur.action;
                  strcpy(last.ciname,cur.ciname);
                  strcpy(last.rel,cur.rel);
                  sprintf(line,"%s %s %s",(*sp).ciname, (*sp).rel, nline);
                  fputs(line,outptr);
                  continue;
            }
            }
        fprintf(stderr,"\n*********** ERROR ***********\n");
        fprintf(stderr," include/exclude entry invalid.\n");
        fprintf(stderr,"entry = %s\n",line);
        fprintf(stderr,"**********************************\n");
        stat = 16;
        continue;
            }
        }
fclose(inptr);
fclose(outptr);
if (stat == 16)
   {fprintf(stderr,"\n*********** ERROR ***********\n");
    fprintf(stderr," execution terminated due to errors.\n");
    fprintf(stderr,"**********************************\n");
    return(stat);
   }

/* remove t.srtblist */
rc = system("rm t.srtblist");
if (rc == -1)
```

```
    {fprintf(stderr,"\n***********  WARNING  ************\n");
     fprintf(stderr," execution of 'rm t.srtblist' failed.\n");
     fprintf(stderr,"************************************\n");
    }

/* recreate path for admin argrument list */
strcpy(path,home);          /* HOME */
strcat(path,scm);          /* HOME/scm */
strcat(path,slash);
strcat(path,xargv[0]);          /* HOME/scm/projid/projfile */
strcat(path,slash);
strcat(path,bldir);
strcat(path,slash);
strcat(path,sprefix);
strcat(path,xargv[1]);       /* HOME/scm/projid/projfile/bldir/s.blname */

/* set xargv[0] to point to "admin" */
xargv[0] = admin;

/* set xargv[1] to point to path */
xargv[1] = path;

/* put '-t' prefix on descfile */
a[cnt][0] = '-';
a[cnt][1] = 't';
strcpy(&a[cnt][2],xargv[2]);
xargv[2] = a[cnt];
cnt++;

/* put creator's LOGNAME in the list of authorized users */
a[cnt][0] = '-';
a[cnt][1] = 'a';
strcpy(&a[cnt][2],logname);
xargv[3] = a[cnt];
cnt++;

/* set '-i' and '-n' arguments */
a[cnt][0] = '-';
a[cnt][1] = 'i';
strcpy(&a[cnt][2],"t.blinput");
xargv[xargc++] = a[cnt];
cnt++;
a[cnt][0] = '-';
a[cnt][1] = 'n';
a[cnt][2] = '\0';
xargv[xargc++] = a[cnt];
cnt++;

/* fork to create a child process */
pid = fork();
```

```
if (pid == -1)
   {fprintf(stderr,"Can't service fork request at this time.\n");
    fprintf(stderr,"Try createbl again later.\n");
    return(16);
   }

/* child process => execute SCCS command */
if (pid == 0)
   execv("/usr/bin/admin",xargv);


/* wait in parent process for child to finish */
while ((wpid = wait(&stat)) != pid);
if (stat != 0)
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr,"SCCS admin request failed.\n");
    fprintf(stderr,"**********************************\n");
    return(stat);
   }

/* remove t.blinput */
rc = system("rm t.blinput");
if (rc == -1)
   {fprintf(stderr,"\n***********  WARNING  ************\n");
    fprintf(stderr," execution of 'rm t.blinput' failed.\n");
    fprintf(stderr,"**********************************\n");
   }

return(0);
}


fetchbl(argc,argv)
int argc;
char *argv[];
{
int i, j, rc, pid, wpid, stat, cnt;

char path[MAXPATH], *sptr;
stat = 0;

/* check the number of arguments supplied */
if (argc < 2)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," fetchbl requires at least 2 arguments.\n");
    fprintf(stderr,"**********************************\n");
    return(4);
   }

/* copy argument pointers to SCCS argrument pointer list */
```

```
for (i=0; i<argc ; i++)
    xargv[i] = argv[i];
xargc = argc;

/* check for valid flags in the argument */
i = xargc;
rc = j = 0;
while (--i > 0)
   if (*xargv[++j] == '-')
     {
      sptr = xargv[j]+1;
      switch(*sptr)
      {
       case 'e':
          /* check authorization */
          if (strcmp(owner,logname) != 0)
             {fprintf(stderr,"\n***********  ERROR  ***********\n");
              fprintf(stderr," This function is retricted for \n");
              fprintf(stderr," use to the administrator.\n");
              fprintf(stderr,"*********************************\n");
              return(256);
             }
         break;
       case 'r':
         break;
       default:
         fprintf(stderr,"\n***********  ERROR  ***********\n");
         fprintf(stderr," Invalid '-' option encountered\n");
         fprintf(stderr,"*********************************\n");
         rc = 8;
         break;
     }
   }
if (rc != 0)
   return(rc);

/* create path for admin argrument list */
strcpy(path,home);          /* HOME */
strcat(path,scm);          /* HOME/scm */
strcat(path,slash);
strcat(path,xargv[0]);          /* HOME/scm/projid/projfile */
strcat(path,slash);
strcat(path,bldir);
strcat(path,slash);
strcat(path,sprefix);
strcat(path,xargv[1]);       /* HOME/scm/projid/projfile/bldir/s.blname */

/* set xargv[0] to point to "get" */
xargv[0] = get;
```

```
/* set xargv[1] to point to path */
xargv[1] = path;


/* fork to create a child process */
pid = fork();
if (pid == -1)
   {fprintf(stderr,"Can't service fork request at this time.\n");
    fprintf(stderr,"Try fetchbl again later.\n");
    return(16);
   }

/* child process => execute SCCS command */
if (pid == 0)
   execv("/usr/bin/get",xargv);

/* wait in parent process for child to finish */
while ((wpid = wait(&stat)) != pid);
if (stat != 0)
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr,"SCCS get request failed.\n");
    fprintf(stderr,"**********************************\n");
    return(stat);
   }
return(0);
}

updatebl(argc,argv)
int argc;
char *argv[];
{
int i, j, k, rc, pid, wpid, stat, cnt;

char path[MAXPATH], *sptr, cmnt[200];

/* check authorization */
if (strcmp(owner,logname) != 0)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," This function is retricted for \n");
    fprintf(stderr," use to the administrator.\n");
    fprintf(stderr,"**********************************\n");
    return(256);
   }
stat = 0;

/* check the number of arguments supplied */
if (argc < 2)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," updatebl requires at least 2 arguments.\n");
    fprintf(stderr,"**********************************\n");
```

```
    return(4);
   }

/* copy argument pointers to SCCS argrument pointer list */
for (i=0; i<argc ; i++)
    xargv[i] = argv[i];
xargc = argc;

/* check for valid flags in the argument */
i = xargc;
rc = j = 0;
while (--i > 0)
   if (*xargv[++j] == '-')
     {
      sptr = xargv[j]+1;
      switch(*sptr)
      {
       case 'y':
         k = j;
         strcpy(cmnt,xargv[j]);
         while (--i > 0)
           {
            strcat(cmnt,blank);
            strcat(cmnt,xargv[++k]);
            xargc--;
           }
         xargv[j] = cmnt;
         xargv[++j] = NULL;
         break;
       default:
         fprintf(stderr,"\n***********  ERROR  ***********\n");
         fprintf(stderr," Invalid '-' option encountered\n");
         fprintf(stderr,"*********************************\n");
         rc = 8;
         break;
      }
   }
if (rc != 0)
   return(rc);

/* create path for admin argrument list */
strcpy(path,home);          /* HOME */
strcat(path,scm);          /* HOME/scm */
strcat(path,slash);
strcat(path,xargv[0]);          /* HOME/scm/projid/projfile */
strcat(path,slash);
strcat(path,bldir);
strcat(path,slash);
strcat(path,sprefix);
strcat(path,xargv[1]);       /* HOME/scm/projid/projfile/bldir/s.blname */
```

```
/* set xargv[0] to point to "delta" */
xargv[0] = delta;

/* set xargv[1] to point to path */
xargv[1] = path;


/* fork to create a child process */
pid = fork();
if (pid == -1)
   {fprintf(stderr,"Can't service fork request at this time.\n");
    fprintf(stderr,"Try updatebl again later.\n");
    return(16);
   }

/* child process => execute SCCS command */
if (pid == 0)
   execv("/usr/bin/delta",xargv);

/* wait in parent process for child to finish */
while ((wpid = wait(&stat)) != pid);
if (stat != 0)
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr,"SCCS delta request failed.\n");
    fprintf(stderr,"**********************************\n");
    return(stat);
   }
return(0);
}

listbl(argc,argv)
int argc;
char *argv[];
{
int i, j, rc, pid, wpid, stat, cnt, cflag, dflag;

char path[MAXPATH], *sptr, argstr[300], l[3], e[3], a[3];
stat = 0;
cflag = dflag = 0;

/* check the number of arguments supplied */
if (argc < 2)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," listbl requires at least 2 arguments.\n");
    fprintf(stderr,"**********************************\n");
    return(4);
   }

/* copy argument pointers to SCCS argrument pointer list */
for (i=0; i<argc ; i++)
```

```
    xargv[i] = argv[i];
xargc = argc;

/* check for valid flags in the argument */
i = xargc;
rc = j = 0;
while (--i > 0)
   if (*xargv[++j] == '-')
     {
      sptr = xargv[j]+1;
      switch(*sptr)
      {
       case 'c':
         cflag = 1;
         break;
       case 'd':
         dflag = 1;
         break;
       default:
         fprintf(stderr,"\n***********  ERROR  ************\n");
         fprintf(stderr," Invalid '-' option encountered\n");
         fprintf(stderr,"**********************************\n");
         rc = 8;
         break;
      }
   }
if (rc != 0)
   return(rc);

/* create path for admin argrument list */
strcpy(path,home);          /* HOME */
strcat(path,scm);          /* HOME/scm */
strcat(path,slash);
strcat(path,xargv[0]);          /* HOME/scm/projid/projfile */
strcat(path,slash);
strcat(path,bldir);               /* HOME/scm/projid/projfile/bldir */
if (*xargv[1] != '.')
   {
    strcat(path,slash);
    strcat(path,sprefix);
    strcat(path,xargv[1]);     /* HOME/scm/projid/projfile/bldir/s.blname */
   }

/* set xargv[0] to point to "prs" */
xargv[0] = prs;

/* set xargv[1] to point to path */
xargv[1] = path;

/* determine output required */
```

```
argstr[0] = '-';
argstr[1] = 'd';
if (cflag == dflag)
    cflag = dflag = 1;
if (dflag == 1)
   {
    strcpy(&argstr[2],bldprs);
    xargv[2] = argstr;
    xargv[3] = NULL;
    xargc = 3;
    /* fork to create a child process */
    pid = fork();
    if (pid == -1)
       {fprintf(stderr,"Can't service fork request at this time.\n");
        fprintf(stderr,"Try listbl again later.\n");
        return(16);
       }
    /* child process => execute SCCS command */
    if (pid == 0)
       execv("/usr/bin/prs",xargv);

    /* wait in parent process for child to finish */
    while ((wpid = wait(&stat)) != pid);
    if (stat != 0)
       {fprintf(stderr,"\n************  ERROR  ************\n");
        fprintf(stderr,"SCCS prs request failed.\n");
        fprintf(stderr,"**********************************\n");
        return(stat);
       }
   }
if (cflag == 1)
   {
    l[0] = '-';
    l[1] = 'l';
    l[2] = '\0';
    e[0] = '-';
    e[1] = 'e';
    e[2] = '\0';
    a[0] = '-';
    a[1] = 'a';
    a[2] = '\0';
    if (dflag == 1)
       strcpy(&argstr[2],blcprs);
    else strcpy(&argstr[2],blcprs2);
    xargv[2] = argstr;
    xargc++;
    xargv[3] = l;
    xargc++;
    xargv[4] = e;
    xargc++;
```

```
    xargv[5] = a;
    xargv[6] = NULL;
    xargc = 6;
    /* fork to create a child process */
    pid = fork();
    if (pid == -1)
       {fprintf(stderr,"Can't service fork request at this time.\n");
        fprintf(stderr,"Try listbl again later.\n");
        return(16);
       }
    /* child process => execute SCCS command */
    if (pid == 0)
       execv("/usr/bin/prs",xargv);
    /* wait in parent process for child to finish */
    while ((wpid = wait(&stat)) != pid);
    if (stat != 0)
       {fprintf(stderr,"\n************  ERROR  ************\n");
        fprintf(stderr,"SCCS prs request failed.\n");
        fprintf(stderr,"**********************************\n");
        return(stat);
       }
   }
return(0);
}

rbldesc(argc,argv)
int argc;
char *argv[];
{
int i, j, k, rc, pid, wpid, stat, cnt;

char path[MAXPATH], *sptr, a[100];

/* check authorization */
if (strcmp(owner,logname) != 0)
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr," This function is retricted for \n");
    fprintf(stderr," use to the administrator.\n");
    fprintf(stderr,"**********************************\n");
    return(256);
   }
stat = 0;

/* check the number of arguments supplied */
if (argc < 3)
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr," rbldesc requires at least 3 arguments.\n");
    fprintf(stderr,"**********************************\n");
    return(4);
   }
```

```
/* copy argument pointers to SCCS argrument pointer list */
for (i=0; i<argc ; i++)
    xargv[i] = argv[i];
xargc = argc;

/* create path for admin argrument list */
strcpy(path,home);          /* HOME */
strcat(path,scm);          /* HOME/scm */
strcat(path,slash);
strcat(path,xargv[0]);          /* HOME/scm/projid/projfile */
strcat(path,slash);
strcat(path,bldir);
strcat(path,slash);
strcat(path,sprefix);
strcat(path,xargv[1]);       /* HOME/scm/projid/projfile/bldir/s.blname */

/* set xargv[0] to point to "admin" */
xargv[0] = admin;

/* set xargv[1] to point to path */
xargv[1] = path;

/* put "-t" prefix on descfile */
a[0] = '-';
a[1] = 't';
strcpy(&a[2],xargv[2]);
xargv[2] = a;


/* fork to create a child process */
pid = fork();
if (pid == -1)
   {fprintf(stderr,"Can't service fork request at this time.\n");
    fprintf(stderr,"Try rbldesc again later.\n");
    return(16);
   }

/* child process => execute SCCS command */
if (pid == 0)
   execv("/usr/bin/admin",xargv);

/* wait in parent process for child to finish */
while ((wpid = wait(&stat)) != pid);
if (stat != 0)
   {fprintf(stderr,"\n************  ERROR  ************\n");
    fprintf(stderr,"SCCS admin request failed.\n");
    fprintf(stderr,"**********************************\n");
    return(stat);
   }
return(0);
```

```
}

blvdesc(argc,argv)
int argc;
char *argv[];
{
int i, j, k, rc, pid, wpid, stat, cnt;

char path[MAXPATH], *sptr, cmnt[200];

/* check authorization */
if (strcmp(owner,logname) != 0)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," This function is retricted for \n");
    fprintf(stderr," use to the administrator.\n");
    fprintf(stderr,"************************************\n");
    return(256);
   }
stat = 0;

/* check the number of arguments supplied */
if (argc < 3)
   {fprintf(stderr,"\n***********  ERROR  ************\n");
    fprintf(stderr," blvdesc requires at least 3 arguments.\n");
    fprintf(stderr,"************************************\n");
    return(4);
   }

/* copy argument pointers to SCCS argrument pointer list */
for (i=0; i<argc ; i++)
    xargv[i] = argv[i];
xargc = argc;

/* check for valid flags in the argument */
i = xargc;
rc = j = 0;
while (--i > 0)
   if (*xargv[++j] == '-')
     {
      sptr = xargv[j]+1;
      switch(*sptr)
      {
       case 'y':
         k = j;
         strcpy(cmnt,xargv[j]);
         while (--i > 0)
           {
            strcat(cmnt,blank);
            strcat(cmnt,xargv[++k]);
            xargc--;
```

```
            }
          xargv[j] = cmnt;
          xargv[++j] = NULL;
          break;
        case 'r':
          break;
        default:
          fprintf(stderr,"\n************  ERROR  ************\n");
          fprintf(stderr," Invalid '-' option encountered\n");
          fprintf(stderr,"************************************\n");
          rc = 8;
          break;
       }
    }
if (rc != 0)
   return(rc);

/* create path for admin argrument list */
strcpy(path,home);          /* HOME */
strcat(path,scm);          /* HOME/scm */
strcat(path,slash);
strcat(path,xargv[0]);          /* HOME/scm/projid/projfile */
strcat(path,slash);
strcat(path,bldir);
strcat(path,slash);
strcat(path,sprefix);
strcat(path,xargv[1]);       /* HOME/scm/projid/projfile/bldir/s.blname */

/* set xargv[0] to point to "cdc" */
xargv[0] = cdc;

/* set xargv[1] to point to path */
xargv[1] = path;


/* fork to create a child process */
pid = fork();
if (pid == -1)
   {fprintf(stderr,"Can't service fork request at this time.\n");
    fprintf(stderr,"Try blvdesc again later.\n");
    return(16);
   }

/* child process => execute SCCS command */
if (pid == 0)
   execv("/usr/bin/cdc",xargv);

/* wait in parent process for child to finish */
while ((wpid = wait(&stat)) != pid);
if (stat != 0)
```

```
    {fprintf(stderr,"\n************  ERROR  ************\n");
     fprintf(stderr,"SCCS cdc request failed.\n");
     fprintf(stderr,"**********************************\n");
     return(stat);
    }
return(0);
}
```

A CONFIGURATION ITEM AND BASELINE IDENTIFICATION SYSTEM FOR SOFTWARE CONFIGURATION MANAGEMENT

by

WILLIAM H. WILSON, IV.

B.S., Virginia Polytechnic Institute and State University, 1970
M.B.A. University of North Carolina at Greensboro, 1983

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Software Configuration Management (SCM) is one of the relatively new product assurance disciplines which have grown in response to the many software failures of the 1970's. SCM is concerned with the consistent labeling of software elements and with tracking and controlling the evolution of all software elements within a software system throughout its life cycle. The primary goal is the development of software which exhibits product integrity.

A survey of current SCM literature was undertaken. Emphasize was placed on literature which explores the nature of SCM thought and literature which describes the implementation of SCM tools.

A SCM configuration item and baseline identification system was developed which can be used to identify and control the evolution of text oriented configuration items, such as specifications, source code, and data definitions. The system also supports the establishment of configuration baselines at critical points during a configuration's life cycle. Facilities were provided for the identification of individual configuration items and baselines, establishing and enforcing access privileges, maintenance of multiple versions of configuration items and baselines, and controlling configuration item and baseline evolution.

The system was designed to be used by other development tools (e.g., activity specification tools, data definition tools, etc.) as the vehicle through which text data is stored and retrieved.