HO

1517

Bell Laboratories Cover Sheet for Technical Memorandum

The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)

Title: **Use of Slists in a Development Environment**

Date: **January 23, 1980**

Other Keywords: **software administration**

TM: **80-3168-1**

Author(s)	Location	Extension	
T.A. Wendt	**HO 1G-322A**	**7568**	Charging Case: **49408-1** Filing Case: **40324-2**

ABSTRACT

An slist is a file which specifies revision levels for SCCS files. This memorandum presents a development environment built on the use of SCCS, enhanced SCCS tools, and slists.

Source code is controlled in a single public tree of SCCS files. Systems are built from SCCS level specifications, given by slists, and are then provided as "arenas" for use by developers and system testers. Procedures and tools for using slists in development are explained, and a typical development sequence is given.

Pages Text: 7	Other: 10	Total: 17	
No. Figures: 2	No. Tables: 0	No. Refs.: 3	

BELL TELEPHONE LABORATORIES, INC.

TM-80-3168-1

DISTRIBUTION
(REFER GEI 13.9-3)

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES

OFFICIAL FILE COPY
PLUS ONE COPY FOR
EACH ADDITIONAL FILING
CASE REFERENCED

DATE FILE COPY
(FORM E-1328)

10 REFERENCE COPIES

BAUER,H C
BEAUMONT,LELAND R
+BONANNI,L E
BREITHAUPT,A R
CICON,J P
DICKMAN,BERNARD N
DONOHUE,E P,3RD
DOUGHERTY,H J
+FELDMAN,STUART I
FREEMAN,K G
GILLON,ALEX C
+GLASSER,ALAN L
HAISCH,H P,JR
HAYDEN,DONALD P,JR
HEIDER,BRUCE H
HERGENHAN,C E
JAASMA,E G
JONES,E E
KAPLAN,FRANK
MACRI,P P
MALIK,JOSEPH M
MARIONE,JOHN P
MAUNSELL,H I
MC DONALD,J P
MCCULLOCH,JOHN W
MITCHELL,J C
MURPHY,LAWRENCE E
OMOHUNDRO,WAYNE E
PASTERNACK,G
PILLA,M A
REBERRY,D L
RODRIGUEZ,ERNESTO J
SHAER,N R
SNYDER,BERNARD E
STREETER,LYNN A
STUBBLEFIELD,R W
TABLOSKI,THEODORE F,JR
TING,DENNIS WAY
WATSON,D S
WELT,M J
WONSIEWICZ,B C
YORK,H L
ZISLIS,PAUL M
43 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

AAGESEN,JOHN
ACKERMAN,J T
ACKROFF,JOHN M
AHO,ALFRED V
AHRENS,RAINER B
ALBERALLA,RICHARD J
ALCALAY,D
ANDERSON,KATHRYN J
ANDERSON,MILTON M
ANTOLICK,DAVID E
APPULINGAM,SUBRAMANIAM
ARNDT,DENNIS L
ARNOLD,GEORGE W
ARNOLD,JAMES Q
ARNOLD,PHYLLIS A
ARNOLD,THOMAS F
ASTHANA,ABHAYA
BABECKI,GLENN E
BABU,RAJESH RATILAL
BAILY,DAVID E
BAIN,WILLIAM LAMAR,JR
BALENSON,CHRISTINE M
BALLANCE,ROBERT A
BARBATO,ROBERT E
BARCLAY,DAVID K
BAROFSKY,ALLEN
BARON,ROBERT V
BAUER,BARBARA T
BAUER,HELEN A
BAUMAN,STEVEN M
BEBLO,WILLIAM
BECKER,CURTIS A
BECKER,RICHARD A
BECK,WANDA
BEDNAR,JOSEPH A,JR
BEGLEY,ALOYSIUS A
BENCO,DAVID S
BENISCH,JEAN
BENNETT,RAYMOND W
BENNETT,RICHARD L
BENNETT,WILLIAM C
BERGLAND,G D
BERNHARDT,RICHARD C
BERNSTEIN,L
BERRYMAN,R D
BERZINS,ALEXANDER H
BILOWOS,R M
BIREN,IRMA B
BITTNER,B B
BLAKE,GARY D
BLAZIER,S D
BLECHMAN,RONALD I
BLZIER,JOSEF
BLINN,J C
BLISS,ROBERT H
BLOSSER,PATRICK A
BLUMER,THOMAS P

COVER SHEET ONLY TO

BLUM,MARION
BODEN,P J
+BOEHM,EARL W
BOEHM,KIM E
BOGART,THOMAS G
BOIVIE,RICHARD H
BOLSKY,MORRIS I
BOPP,J ROBERT
BORDELON,EUGENE P
BORISON,ELLEN A
BOURNE,STEPHEN E
BOWYER,L RAY
BOYCE,W M
BOYER,PHYLLIS J
BOYLE,GERALD C
BRADLEY,M HELEN
BRADLEY,E H
BRADY,PAUL T
BRIGGS,GLORIA A
BROAD,MARTHA M
BRONSTEIN,N
BRONZO,JOSEPH A
BROOKS,CATHERINE ANN
BROSS,JEFFREY D
BROWMAN,INNA
+BROWN,D W
BROWN,ELLINGTON L
BROWN,LAURENCE MC FEE
BROWN,W STANLEY
BRUECKNER,DOUGLAS E
BURG,F M
BURKE,THERESA M
BURNS,GARY J
BUROFF,STEVEN J
BUSCH,KENNETH J
BUTLETT,DARRELL L
BYERLEE,E W
BYCHICK,ROBERT S
BYRNE,EDWARD E
CAMPBELL,JERRY H
CANADAY,RUDD H
CANDREA,RONALD D
CAREY,F E,JR
CARTER,DONALD H
CASPERS,BARBARA E
CASTELLANO,MARY ANN
CATO,H E
CAVINESS,JOHN D
CERMAK,I A
CHAI,D T
CHAMBERS,B C
CHAMBERS,J M
CHANG,CHUNG W
CHANG,JO-MEI
CHAPPELL,S G
CHENG,Y
CHEN,ROBERT
CHEN,STEPHEN
CHERRY,LORINDA L
CHIEN,FU-LI BETTY
CHILDS,CAROLYN
CHIN,KATHLEEN E

COVER SHEET ONLY TO

CHI,M C
CHODROW,M M
CHONG,PHEE
CHRISTENSEN,S W
CHRIST,C W,JR
CIEMINSKI,DEBRA F
CLARK,DAVID L
CLAYTON,D P
CLEWELL,JAMES L
CLINE,LAUREL M I
COHEN,DAVID
COHEN,HARVEY
COHEN,KAREN L
COHEN,NEIL B
COHEN,RICHARD L
+COLBY,ROBERT W
<COLE,LOUIS M
COLLICOTT,R B
COLWELL,ELLEN MARIE
CONKLIN,DANIEL L
CONNERS,RONALD R
COOK,JOEL M
COOK,T J
COOPER,ARTHUR E
COSCHIGANO,J J,JR
COSTON,W P
COTTRELL,JENNIE L
CRAIG,JOHN R
CRISTOPOR,EUGENE
CRUPI,JOSEPH A
DAVISON,JOSEPH W
DAVIS,E DREW
DAY,F W
DE FAZIO,M J
DE GRAAF,D A
DE TREVILLE,JOHN D
DEAN,JEFFREY S
DENNY,MICHAEL S
DENSMORE,SUSAN
DESMOND,JOHN PATRICK
DEVLIN,SUSAN J
DI PIETRO,R S
DIB,GILBERT
DIMMICK,JAMES O
DINEEN,THOMAS J
DOCK,G A
DOEDLINE,BARBARA ANN
DOLATOWSKI,VIRGINIA M
DOLOTTA,T A
DOMANSKI,BERNARD
DOWDEN,DOUGLAS C
DOWDEN,IRIS S
DOYLE,C
DRAKE,LILLIAN
DREIZLER,H K
D'ANDREA,LOUISE A
DUDICK,ANTHONY
DUGGER,DONALD D
DUMAIS,VALERIE
DUNCANSON,ROBERT L
DUNKIN,PATRICIA
DURAND,JANIS C

COVER SHEET ONLY TO

DWYER,T J
DYER,MARY E
EICHORN,KURT H
EISELE,RONALD C
EITELBACH,DAVID L
EKSTROM,SUSAN
ELDRIDGE,JOHN
+ELLIOTT,WILLIE
ELLIS,DAVID J
ELI,T C
EMERSON,DOMINIQUE M
ENGLAR,BRUCE WYATT
EPLEY,ROBERT V
ESCOLAR,CARLOS
ESSERMAN,ALAN R
FABISCH,M P
FABRICIUS,WAYNE N
FAIRCHILD,DAVID L
FALLON,J M
FAULKNER,ROGER A
+FEAY,MARY R
FEDER,J
FELTON,WILLIAM A
FERRER,NANCY L
FEUSTER,I REED
FISCHER,HERBERT B
FISHER,DIDINA
FISHMAN,DANIEL H
FLANDRENA,R
FLEISCHER,H I
+FLYNN,EVELYN
FORSON,HENRY M
FORTNEY,V J
FOUGHT,B T
FOUNTOUKIDIS,A
FOWLER,BRUCE B
FOWLER,GLENN D
FOX,PHYLLIS A
FOY,J C
FRANKLIN,JAMES W
FRANK,AMALIE J
FRASER,A G
FREEMAN,MARTIN
FREMON,R C
FRIED,LAWRENCE K
FROST,H BONNELL
FRUCHTMAN,BARRY
GABBE,JOHN D
GALE,ALAN O
GANA,JORGE L
GARRISON,GARY A
GARST,BLAINE,JR
GATES,G W
GAWRON,L J
GEERS,T J,JR
GEORGEN,MICHAEL E
GEPNER,JAMES R
GEILING,F T
GIBB,KENNETH R
GIBSON,J C
GIFFORD,WARREN S
GILBERT,ROBERT
...
740 TOTAL

+ NAMED BY AUTHOR > CITED AS REFERENCE < REQUESTED BY READER (NAMES WITHOUT PREFIX WERE SELECTED USING THE AUTHOR'S SUBJECT OR ORGANIZATIONAL SPECIFICATION AS GIVEN BELOW)

MERCURY SPECIFICATION.....

COMPLETE MEMO TO:
316-SUP

COVER SHEET TO:
316-TA

COPRPM = COMPUTER PROGRAMMING MANAGEMENT

---

nO CORRESPONDENCE FILES
HO 1A-127

TM-80-3168-1
TOTAL PAGES 17

PLEASE SEND A COMPLETE

( ) MICROFICHE COPY ( ) PAPER COPY

TO THE ADDRESS SHOWN ON THE OTHER SIDE.

TO GET A COMPLETE COPY:

1. BE SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.
2. FOLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.
3. CIRCLE THE ADDRESS AT RIGHT. USE NO ENVELOPE.
4. INDICATE WHETHER MICROFICHE OR PAPER IS DESIRED.

Bell Laboratories

subject: **Use of Slists in a Development Environment**
Case: **49408-1**
File: **40324-2**

date: **January 23, 1980**

from: **T.A. Wendt**
HO 3168
1G-322A x7568

TM: **80-3168-1**

ABSTRACT

An slist is a file which specifies revision levels for SCCS files. This memorandum presents a development environment built on the use of SCCS, enhanced SCCS tools, and slists.

Source code is controlled in a single public tree of SCCS files. Systems are built from SCCS level specifications, given by slists, and are then provided as "arenas" for use by developers and system testers. Procedures and tools for using slists in development are explained, and a typical development sequence is given.

MEMORANDUM FOR FILE

1. Introduction

The development of a software product is not accomplished in a single step. Numerous revisions to source files are common. If the software product consists of a large system, it is not unusual for multiple versions of the system - produced from differing editions of many source files - to be in use simultaneously. PWB/UNIX[1] provides a number of tools to assist with version control; chief among these are *make* [1] and the SCCS [2] system. A group of additional tools has been developed, as part of the Advanced Communications Service (ACS) effort, to extend the SCCS system of control. These tools confer the ability to tie together the version control information for an entire software package into a single file, which is stored under SCCS. The SCCS version number of this file becomes the version number of the entire software package. Given the SCCS-controlled source, then, a developer or administrator can easily re-make a complex software product - consisting of many separate files - at any desired version.

Each SCCS file is a composite of multiple versions (from 1 to very many) of a source file. Each version is identified by a number. Taken together, the name of a source file (e.g., **x.c**) and one of its version numbers (e.g., **1.3**) comprise an SCCS id (e.g., **x.c 1.3**). The SCCS command *get* can be used to extract particular versions of files and, in combination with the SCCS command *delta*, create new versions.

Groups of SCCS id's can be placed in a single file called an slist (SCCS id list). For example, versions of files **c1.c**, **c2.c**, and **c3.c** might be represented in an slist as shown in figure 1.

1. UNIX is a Trade/Service Mark of the Bell System.

```
c1.c    1.4
c2.c    2.5
c3.c    1.3
```

Figure 1. A simple slist

Slists are designed to be used with the commands *sget*, *sdelta*, and *chkvers*[2], forming a simple software revision control system [3]. Within the scope of software development, this system is used by associating one slist with one software "object" - a process, a library, or a header file directory - and using the slist to specify that object when building a software release.

Each slist can completely specify each object, because an slist may be produced by executing the SCCS *what* command on an object. *What* extracts the SCCS id's of all the files used in making the object, including modules not explicitly created by the developer but only referenced (e.g., header files incorporated via the *C*

facility, or library modules incorporated by the loader). The output from *what* can be collected in a file, as in

```
what xxx > Sl.xxx
```

producing a complete slist for module **xxx**. Completeness produces useful redundancy, as slists may then be compared to assure the compatibility of common components (e.g., revision levels of a common header file should be identical in all slists making up a release). This memorandum describes a development system based on the use of slists.

2. A Development Environment

In our development system, a software release can be viewed as passing through three different groups as it moves from a set of requirements to a product in the field. In order of passage, these are

1. developers, including those who design, code, and perform initial stand-alone and system tests;
2. the Software Integration group, which administers the public source code and the load modules (in a collection called an "arena") produced from that source as specified by the developers; and
3. the System Test group, which produces the final arena, tests it thoroughly and, if it passes the tests, certifies it as fit for release to the field.

Slists - software specifications - not only form the basis for communication among the above groups, but are a regular part of the daily development routine.

2.1 Components of the Development Environment

There are three primary components to the development environment:

1. The SCCS source tree - the directory structure in which the SCCS-controlled version of official software is kept,
2. The current arena - the current version of official source and the corresponding load modules which have been compiled, tested, and found to be internally consistent by the Software Integration group; and
3. The developer's local directory structure - the work-space in which the developer's private version of software is kept prior to its inclusion in an arena.

2. See attached manual pages.

2.2 The SCCS Tree

Creating a single directory tree in which all source files reside is an attempt to gain control over the creation and maintenance of those files. Our objectives included

1. maintaining a complete history of changes for each source file,
2. making the source code public and available for inspection by any developer or administrator,
3. imposing the logical structure of the project on the organization of source files, and
4. providing the capability of automatically checking and enforcing coding standards.

With the establishment of the project SCCS tree, all source files are collected - as SCCS files - under a single UNIX directory structure. This tree exists in a well-known place, and all files (which are uniquely-named) and directories within it are readable by everyone. The SCCS tree is controlled by the Software Integration group to the extent that

- only members of the Software Integration group may create or delete files or directories in the tree, and
- permission to apply SCCS deltas to files in the SCCS tree must be given to a developer by the Software Integration group.

A developer with "delta permission" may use SCCS (through *sget* and *sdelta*) to make any number of changes to a source file.

Our first two objectives are immediately met with the creation of the SCCS tree: the use of SCCS automatically causes retention of change history, and the use of a single directory structure, in which all files are generally readable, makes the source code public.

Our other two objectives are met through the procedures used to administer the SCCS tree. Because only Software Integration may add or delete files or directories in the tree, they are in a position to

1. control the placement of files which they create at developers' requests, and
2. enforce coding standards by rejecting files which do not meet those standards.

The SCCS tree may be designed so that processes which are functionally related are close to each other in the tree, and common modules (e.g., libraries, header files) are positioned close to the root of the subtree which uses them. This allows an easier grasp of the structure of the software system as a whole, and allows developers to concentrate on a single subtree for their own purposes.

The Software Integration group does not create empty files in the SCCS tree to allow the developer to fill in the contents of the file at some later time. Initial versions of source code files are created in a developer's private directory; the SCCS version in the tree is created from that private version. Automating the procedure allows a program, run on behalf of the Software Integration group, to check for adherence to coding standards.

2.3 The Arena

An arena is a snapshot of the SCCS tree. It is a collection of the "current" versions of source files plus the load modules and libraries which are made from that source. The directory structure of the arena mimics that of the SCCS tree; the location of the SCCS version of a file in the tree dictates the location of the extracted version of that file in the arena.

The arena is produced by Software Integration using only files from the SCCS tree. Developers specify their software to Software Integration through slist revision levels. After accepting new slist levels, the Software Integration group extracts from the SCCS tree each slist and the files it specifies, then executes each makefile (listed along with other source files in the slist) to produce processes and libraries. The Software Integration group assures that the slist

versions specify modules which

1. are consistent across the arena, each source file being represented in only one version;
2. compile without error; and
3. meet some minimum tests of execution.

From the developer's standpoint, the completed arena serves as the official version of common (i.e., used by more than one process) software. Header files and libraries maintained by other developers may be found in well-known places inside the arena. Further, they are not subject to arbitrary change: an arena is stable after its creation.

2.4. Local Directory Structure

Each developer, when building a process, works in a private uncontrolled directory structure away from the arena. The private directory structure should be a copy of that subtree of the SCCS structure with which the developer or group of cooperating developers is directly concerned. It is important to differentiate between the developer's files in this private directory structure (which he or she can change) and those in the arena (which are fixed for the life of the arena).

Makefile conventions have been written to simplify the transition between the arena directory structure and the local directory structure. Common header files (i.e., those which are referenced by multiple processes) are referenced with pathnames prefaced by the **ARENA** macro. All local files are referenced relative to the working directory. Thus a header file **header.h** would be referenced as

```
../hd/header.h
    or
$(ARENA)/$(MYPATH)/hd/header.h
```

depending on whether it was local or not. (**ARENA** and **MYPATH** are *make* macro definitions: **ARENA** would be defined as the path to the top of the arena directory structure; **MYPATH** would be a path through a subtree of the arena.) These definitions work whether the makefile is executed by the developer in a private directory, or by Software Integration inside the arena.

As an example, consider the development of some complex part of a software system. Assuming there are numerous source files, it is likely that these source files may be partitioned into logical sets[3]. This source code partitioning is performed by placing source files into more than one directory in the SCCS tree, with all header files placed into an additional directory.

In order to manipulate the source files for the above subsystem, a developer would first create a local directory structure which matches the SCCS subtree in which the source for the subsystem is stored. Figure 2 shows a possible structure. The subtree headed by the directory **process** should match a subtree headed by a directory **process** in the SCCS tree. When building the example subsystem, the developer first gets specific versions of source files from the SCCS subtree into this local subtree, then processes (e.g., compiles) these local files.

3. Tools for Use With Slists

Modified versions of the SCCS *get* and *delta* commands have been created for use with slists. These commands - called *sget* and *sdelta* - perform the same functions with respect to SCCS files as do *get* and *delta*. In addition, they use and manipulate slists: *sget* reads the slist to obtain the SCCS version number to be used when performing a *get*, and *sdelta* automatically updates the slist with the new SCCS version number when performing a *delta*. As an added feature,

3. Cause for such partitioning might arise in a subsystem which incorporates two distinct functions. A debugging scheme could be developed in which each function would be coded in a separate group of source files, with each group producing a separate partial load module. The production load module would be produced by combining the two partial modules: debugging of each separately could be achieved by substituting a special debugging module for the other load module.

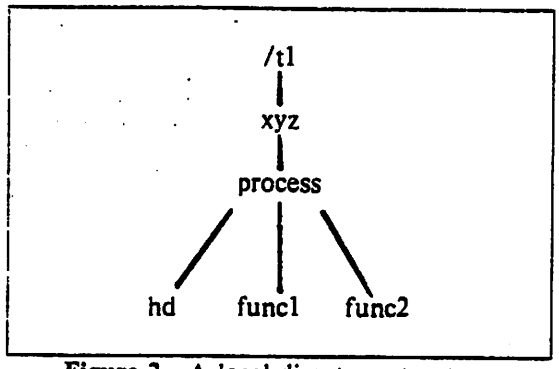

Figure 2. A local directory structure

both *sget* and *sdelta* are designed to do controlled path searching: when accessing files in the SCCS tree, only the file name need be given - not the entire path into the tree. A file containing directory pathnames (called a "dirlist") is used to control the searching.

The use of slists is built around the following properties of *sget* and *sdelta*:

1. Both *sget* and *sdelta* will search for an slist and a dirlist (both are required by default) in the current directory and the immediate parent directory.
2. *Sget* will perform a *get* of named files or (default) all files in the slist from a SCCS directory (specified by the dirlist) into the current directory. Unless directed to do otherwise, it will handle a multi-line dirlist by assuming isomorphism between the SCCS tree and the local directory structure. Only one directory will be searched - the one whose entry in the dirlist best matches the pathname of the working directory (matching against the last component first, then the next-to-last, etc.).
3. *Sget* will get files at the version specified by the first slist it finds.

4. Procedures for Using Slists

The following are general guidelines for slist use. The subsystem example shown previously will be used for illustration. The term **SCCSTREE** will be used to stand for the UNIX pathname to the root of the SCCS tree.

1. Each local directory should contain a dirlist containing the path to the SCCS directory from which files for this directory will be obtained.

   For example, the **/t1/xyz/process/hd** directory will contain a file **DI.** whose contents will be

   SCCSTREE/process/hd

2. Slists should be placed in the directory which is the immediate parent of directories containing source files. This will allow the slist to be found automatically by *sget* and *sdelta*. There should be only one slist in any directory, to avoid errors.

   As an example, consider an slist for our example subsystem. Local files exist in directories under **SCCSTREE/process**. The slist should be placed in directory **SCCSTREE/process**. A local copy of this slist would be brought into directory **/t1/xyz/process**.

3. All source files must include the SCCS keyword "%W%". C language source, for example, should include the line

   static char xxxx[] = "%W%";

   (where **xxxx** is some unique variable name - e.g., an abbreviation of the file name) as the first line of every file[4]. This ensures the proper functioning of the file control tools.

4. To conserve main memory, we have found that compilers, assemblers, and loaders can be modified to allow SCCS

4. Modifications to source files should be done while a local copy of the slist is available for editing: the slist should have been gotten using the -e flag. This will allow an updated slist to be put back (via *sdelta*) into SCCS when a completed version of the product has been achieved. This version can then be regenerated from SCCS files.

5. Each makefile should automatically produce an slist which is a *what* of the load module or library produced by the makefile. This slist should also contain the SCCS id of the makefile in order to fully specify the load module. The simplest method of making this slist is through repeated executions of the *what* command. Appendix A gives a detailed explanation of a set of tools and makefile actions which produce a proper slist and, in addition, incorporate the slist's SCCS id into the final load module where it may be found in memory dumps.

   A simple slist may be created for our subsystem as follows. Assuming that the subsystem is contained in one load module (discounting possible intermediate load steps) called **a.out**, and the makefile which makes this load module is **make.process**, we can create an slist by appending the following lines to the makefile after the line in which **a.out** is created:

   ```
   what a.out > NSl.process
   what make.process >> NSl.process
   ```

   The first line creates an slist which is a *what* of the load module, and the second appends a *what* of the makefile. The slist (temporarily called **NSL.process**) must be edited to include the "%W%" keyword.

6. The developer should execute *chkvers* on the new and old slists. The *chkvers* program will compare two slists and report their differences. Unexpected differences indicate potential problems.

5. The Development Cycle

Given the development environment and tools outlined above, a picture of typical software development can be given. This development covers the span from completed design to inclusion of the official version of the software into an arena, and thence to the incorporation of enhanced software versions in later arenas.

The development process can be broken into stages. Passing through those stages, the software moves from a set of routines which are completely private and known only to the developer, to a stage in which the source code is public but the object is still private, and finally to a stage in which the final product is public and usable by everyone in the project. These stages can be represented as follows:

1. The initial coding stage. Development begins in a private directory structure whose organization is not tied to the SCCS tree. The source code is entirely private.

2. Installation of files in SCCS. When a stable set of source files has been established, they are put into the SCCS tree (for safety):

   a. Software Integration creates a new subtree in the SCCS tree.

   b. Files are installed by Software Integration, after standards check.

   The source code is now public.

3. Development under slist control. With source files now stored under SCCS, it becomes possible (and advisable) to track source code with an slist.

---

id's to be included in comments, from which they may be extracted and treated specially. The SCCS id strings can be made part of the load module (so that a *what* command produces the desired result) but - except for the SCCS id of the slist - not actually loaded.

a. The private directory structure is reorganized to follow the SCCS subtree.

b. An initial slist is made, either with the *mkslist*[5] tool, or (preferably, if possible) through execution of a makefile, as seen above.

c. The slist is installed, and a local copy made by getting it for a delta.

d. As deltas are made to existing source files and new files are put into SCCS, changes are tracked in the slist.

4. Specification of software for an arena. At some point, the module itself will be made public. The developer makes the module, tests it thoroughly, then puts the slist back into SCCS with *sdelta*. He or she then informs Software Integration as to which version of the slist to use in making the module.

5. After creation of the arena, the developer may want to enhance the module. The slist is gotten for a delta, then source files are gotten, modified, *sdelta*ed, and compiled to make a new module and a new slist. After testing, the slist is *sdelta*ed. The new slist revision number can be submitted to Software Integration.

6. Conclusion

A development environment has been presented in which tools and procedures have been created to help in managing software production. Slists and the tools to manipulate them can interact with central SCCS administration of source code to provide reproducible, easily-modified systems.

7. Acknowledgment

The ideas presented in this memorandum are the product of many people, both inside and outside the Software Integration group. In particular, A. L. Glasser is responsible for the concept and implementation of slists, and R. H. Bliss for the makefile conventions in Appendix A. The presentation was greatly enhanced through the suggestions of G. Cristofor and B. C. Wonsiewicz.

T.A. Wendt

HO-3168-TAW-taw

Atts.
References
Appendices A, B

5. See attached manual page.

References

[1] S.I. Feldman, "Make - A Program for Maintaining Computer Programs," TM 75-1274-26, December 5, 1975.

[2] L.E. Bonanni and C.A. Salemi, "Source Code Control System User's Guide," TM 79-9441-4, TM 79-9442-1, April 1, 1979.

[3] A.L. Glasser, "A Simple Software Control System," In preparation, 1979.

Appendix A

Makefile Conventions

A set of conventions and tools have been produced to aid in the generation of makefiles which automatically produce slists. This appendix gives an overview of the use of those tools.

Each makefile has, as its default target, a dummy label (e.g., **all**) which depends on the ultimate load module (or library) and on a separate slist target. The slist target label (e.g., **SL**), in its associated action, makes a new slist by performing a *what* of the load module and adding the "%W%" SCCS keyword (see attached manual page for *Slgen*).

In order to fully specify the process, the makefile version number should appear in the slist. Additionally, the slist version number should appear in the load module where it can be extracted from process dumps. To satisfy these considerations, an object file containing only the SCCS id strings for the makefile and the slist is compiled into the target load module (from which they can be extracted through *Slgen*). This object file can be produced from C source dynamically generated within the makefile (see attached manual page for *makeid*).

The slist formed in this procedure is of the form which is needed by Software Integration. Before the slist can be turned over to Software Integration, it must be put back (via *sdelta*) into the SCCS tree. This implies that, whenever the makefile is executed, the slist has been previously gotten from SCCS for a delta (i.e., with the -e flag).

As an example, consider a possible makefile for the subsystem discussed in the main memorandum. This makefile, executing in the **funcl** directory of the subtree, makes a load module **a.out**. Significant parts of the makefile include:

- the initial target, which depends on the load module and the slist

```
all: a.out SL
```

- the **id.o** file containing SCCS id strings for the makefile and slist. It is always made in every makefile invocation through its dependence on a dummy label **dummy**.

```
id.o: dummy
        makeid make.process ../Sl.process
        cc -c id.c

dummy:
```

  Note that since the makefile executes in the **process/funcl** directory, it references the slist in its parent directory.

- the load step which results in the creation of the desired load module. The load module **a.out** must include the file **id.o**.

```
ld part1 part2 ... id.o
```

  Files **part1**, **part2**, etc., are intermediate load modules or object files.

- the slist generation step, which creates an slist **NSl.process** from the load module **a.out**.

```
SL: a.out
        Slgen Sl.process a.out
```

  Note that the *Slgen* procedure only uses the name **Sl.process** - it does not make reference to the actual slist.

Appendix B

Manual Pages

NAME

Slgen – generate a new slist for load modules

SYNOPSIS

Slgen Sl.name load_module_name

DESCRIPTION

Slgen generates a new slist by running the SCCS command *what*(1) on the specified load modules. A "%W%" is prepended to the resultant file which is left in **NSl.name**. **NSl.name** is in the proper form to be put into the SCCS tree as a 'delta' to **Sl.name**.

NAME

chkvers — check consistency of slists

SYNOPSIS

chkvers new-slist [old-slist]

DESCRIPTION

An slist is the output of the *what*(I) command. It contains names and version numbers of files comprising the load (object) module. *Chkvers* checks the self-consistency of **new-slist**. If different versions of the same file appear in the slist, *chkvers* will write a message on standard error, identifying the name and the version number of the file. If **old-slist** is specified, it also cross-checks the consistency between the old and the new lists. Names of files with different version numbers in the **new-slist** and the **old-slist** are printed on standard output. Names of files which appear in one slist but not the other are also printed. As usual, '-' assumes standard input.

SEE ALSO

what(I)

NAME

makeid − generate a 'C' module called id.c containing the SCCS id's of the files named in the argument list

SYNOPSIS

makeid file [file ...]

DESCRIPTION

Makeid generates a file named **id.c** which contains the SCCS id of each of the files named in the argument list. Normally, arguments to *makeid* are the name of a makefile and the name of an slist. *Makeid* uses the *what*(1) command to generate the SCCS id of each named file, then prepends the string @(#) to each SID so a subsequent *what* of **id.c** will produce the SCCS id's of the files named in the *makeid* call.

SCCS id's are normally placed within C comments in **id.c** where they will not generate loadable code. The SCCS id from an slist (i.e., a filename starting in Sl.) is placed in the initialization field of a static character array, and so generates loadable data.

EXAMPLE

The call

```
makeid make.name Sl.name
```

might produce a file **id.c** containing the following:

```
/*
@(#)make.name        1.3
*/
static char Sl_name [] = "@(#)Sl.name1.1";
```

NAME

mkslist − generate an slist

SYNOPSIS

mkslist [−d] [−h"title"] [dir]

DESCRIPTION

Mkslist is a Shell that will generate an slist of a whole file structure, or a single directory.

−d	generate slist only for the directory, default is for the whole structure
−h"title"	use the string in *title* as the first line of output, instead of the default title
dir	pathname of entry directory, defaults to current directory.

The slist is generated on the standard output, and contains the following header lines:

: Slist of xxx
: %W%

The header lines are followed by the body of the slist proper.

FILES

/tmp/s$$	slist output file
/tmp/$$??	work files for *split*(1)

DIAGNOSTICS

Usage: mkslist [-d] [-h"title"] [dir]
mkslist: bad flag -xxx
mkslist: yyy not directory

EXAMPLES

WARNINGS

RANGES and LIMITATIONS

CAVEAT

The slist generated will contain its own name as well.

SEE ALSO

BUGS

NAME

sdelta − interface to delta

SYNOPSIS

sdelta [−Ddirlist] [−Sslist] [deltaarg] ...

DESCRIPTION

Sdelta is an interface to *delta*(I). Like *sprt*(I) (q.v.), *sdelta* simplifies the amount of typing required by the user by using a *dirlist*. (See *sprt*(I) for a description of the *dirlist* mechanism.)

In addition to the *dirlist*, *sdelta* supports an *slist* file (see *sget*(I) for a description of an *slist*). For each file argument, *sdelta* executes *delta*(I) and, after the delta is made, updates the *slist* with the SID of the delta just made. The *slist* processing can be inhibited by supplying an argument of −S (i.e., a null *slist*).

Typical use is:

```
sdelta file.c
```

FILES

Dl.*	First try for default *dirlist*.
../Dl.*	Second try for default *dirlist*.
dirlist	Third try for default *dirlist*.
../dirlist	Fourth try for default *dirlist*.
Sl.*	First try for default *slist*.
../Sl.*	Second try for default *slist*.
slist	Third try for default *slist*.
../slist	Fourth try for default *slist*.

SEE ALSO

sget(I), sprt(I), delta(I)

A Simple Software Control System by Alan Glasser. In preparation.

NAME

sget − interface to get

SYNOPSIS

sget [−Ddirlist] [−Sslist] [−W] [getarg] ...

DESCRIPTION

Sget is an interface to *get*(I). Like *sprt*(I) (q.v.), *sget* simplifies the amount of typing required by the user by using a *dirlist*. (See *sprt*(I) for a description of the *dirlist* mechanism.)

In addition to the *dirlist*, *sget* supports an *slist* file. An *slist* is the output of a *what*(I) command on a load module. For each file argument to *sget*, the *slist* is searched for that name, and a *get*(I) at the SID specified in the *slist* is executed. The *slist* processing can be inhibited by supplying an argument of −S (i.e., a null *slist*).

The −W option will inhibit the *getting* of a file if it exists and is at the proper SID (as determined by an internal *what* of the file). This option can be used to minimize recompilation by not changing the modifided time of the file unnecessarily, and thereby not cause *make*(I) to remake the object file.

Typical use is:

```
sget file.c
```

or

```
sget −e file.c
```

FILES

Dl.*	First try for default *dirlist*.
../Dl.*	Second try for default *dirlist*.
dirlist	Third try for default *dirlist*.
../dirlist	Fourth try for default *dirlist*.
Sl.*	First try for default *slist*.
../Sl.*	Second try for default *slist*.
slist	Third try for default *slist*.
../slist	Fourth try for default *slist*.

SEE ALSO

sprt(I), sdelta(I), get(I)

A Simple Software Control System by Alan Glasser. In preparation.

NAME

sprt – interface to prt

SYNOPSIS

sprt [–Ddirlist] prtarg ...

DESCRIPTION

Sprt is an interface to *prt*(I) that simplifies the amount of typing required by the user by using a *dirlist*. A *dirlist* is a file containing the complete pathnames of directories containing SCCS files. The user need only type the simple name (last component) of the SCCS file name (the "s." will be supplied if omitted). *Sprt* transforms its file arguments before passing them to *prt*(I).

Typical use is:

```
sprt file.c
```

where s.file.c is the name of a file in one (or more!) directory(s) listed in the *dirlist*.

FILES

Dl.*	First try for default *dirlist*.
../Dl.*	Second try for default *dirlist*.
dirlist	Third try for default *dirlist*.
../dirlist	Fourth try for default *dirlist*.

SEE ALSO

sget(I), sdelta(I), prt(I)

A Simple Software Control System by Alan Glasser. In preparation.