14. SOURCE CODE CONTROL SYSTEM (SCCS)

Introduction

The Source Code Control System (SCCS) is a maintenance and enhancement tracking tool that runs under the operating system. SCCS takes custody of a file and, when changes are made, identifies and stores them in the file with the original source code and/or documentation. As other changes are made, they too are identified and retained in the file.

Retrieval of the original or any set of changes is possible. Any version of the file as it develops can be reconstructed for inspection or additional modification. History data can be stored with each version: why the changes were made, who made them, when they were made.

This guide covers the following:

- SCCS for Beginners: how to make, retrieve, and update an SCCS file
- Delta Numbering: how versions of an SCCS file are named
- SCCS Command Conventions: what rules apply to SCCS commands
- SCCS Commands: the fourteen SCCS commands and their more useful arguments
- SCCS Files: protection, format, and auditing of SCCS files

Neither the implementation of SCCS nor the installation procedure for SCCS is described in this guide.

SCCS For Beginners

Several terminal session fragments are presented in this section. Try them all. The best way to learn SCCS is to use it.

Terminology

A delta is a set of changes made to a file under SCCS custody. To identify and keep track of a delta, it is assigned an SID (SCCS IDentification) number. The SID for any original file turned over to SCCS is composed of release number 1 and level number 1, stated as 1.1. The SID for the first set of changes made to that

file, that is, its first delta is release 1 version 2, or 1.2. The next delta would be 1.3, the next 1.4, and so on. More on delta numbering later. At this point, it is enough to know that by default SCCS assigns SIDs automatically.

Creating an SCCS File via admin

Suppose, for example, you have a file called **lang** that is simply a list of five programming language names. Use a text editor to create file **lang** containing the following list.

```
C
PL/1
FORTRAN
COBOL
ALGOL
```

Custody of your **lang** file can be given to SCCS using the **admin** command (i.e., administer SCCS file). The following creates an SCCS file from the **lang** file:

```
admin -ilang s.lang
```

All SCCS files must have names that begin with **s.**, hence **s.lang**. The **-i** keyletter, together with its value **lang**, means **admin** is to create an SCCS file and initialize it with the contents of the file **lang**.

The **admin** command replies

```
No id keywords (cm7)
```

This is a warning message that may also be issued by other SCCS commands. Ignore it for now. Its significance is described later with the **get** command under "SCCS Commands." In the following examples, this warning message is not shown although it may be issued.

The **lang** file is no longer needed because it exists now under SCCS as **s.lang**. Remove the **lang** file:

```
rm lang
```

Retrieving a File via get

Use the **get** command as follows:

```
get s.lang
```

This retrieves **s.lang** and prints:

```
1.1
5 lines
```

This tells you that **get** retrieved version 1.1 of the file, which is made up of five lines of text.

The retrieved text has been placed in a new file known as a "g.file." SCCS forms the g.file name by deleting the prefix **s.** from the name of the SCCS file. Thus, the original **lang** file has been recreated.

If you list, **ls**(1), the contents of your directory, you will see both **lang** and **s.lang**. SCCS retains **s.lang** for use by other users.

The **get s.lang** command creates **lang** as read-only and keeps no information regarding its creation. Because you are going to make changes to it, **get** must be informed of your intention to do so. This is done as follows:

```
get -e s.lang
```

get -e causes SCCS to create **lang** for both reading and writing (editing). It also places certain information about **lang** in another new file, called the "p.file" (**p.lang** here), which is needed later by the **delta** command.

get -e prints the same messages as **get**, except that now the SID for the first delta you will create is issued:

```
1.1
new delta  1.2
5 lines
```

Change **lang** by adding two more programming languages:

```
SNOBOL
ADA
```

Recording Changes via delta

Next, use the **delta** command as follows:

```
delta s.lang
```

delta then prompts with:

```
comments?
```

Your response should be an explanation of why the changes were made. For example:

added more languages

delta now reads the p.file, **p.lang**, and determines what changes you made to **lang**. It does this by doing its own **get** to retrieve the original version and applying the **diff**(1) command to the original version and the edited version. Next, **delta** stores the changes in **s.lang** and destroys the no longer needed **p.lang** and **lang** files.

When this process is complete, **delta** outputs:

```
1.2
2 inserted
0 deleted
5 unchanged
```

The number 1.2 is the SID of the delta you just created, and the next three lines summarize what was done to **s.lang**.

Additional Information about get

The command:

get s.lang

retrieves the latest version of the file **s.lang**, now 1.2. SCCS does this by starting with the original version of the file and applying the delta you made. If you use the **get** command now, you can retrieve version 1.2. with any of the following:

get s.lang
get –r1 s.lang
get –r1.2 s.lang

The numbers following **–r** are SIDs. When you omit the level number of the SID (as in **get –r1 s.lang**), the default is the highest level number that exists within the specified release. Thus, the second command requests the retrieval of the latest version in release 1, namely 1.2. The third command specifically requests the retrieval of a particular version, in this case also 1.2.

Whenever a major change is made to a file, you may want to signify it by changing the release number, the first number of the SID. This, too, is done with the **get** command:

get –e –r2 s.lang

Because release 2 does not exist, **get** retrieves the latest version before release 2. **get** also interprets this as a request to change the release number of the new delta to 2, thereby naming it 2.1 rather than 1.3. The output is:

```
1.2
new delta 2.1
7 lines
```

which means version 1.2 has been retrieved, and 2.1 is the version **delta** will create. If the file is now edited (for example, by deleting COBOL from the list of languages), and **delta** is executed:

```
delta s.lang
comments? deleted cobol from list of languages
```

you will see by **delta**'s output that version 2.1 is indeed created:

```
2.1
0 inserted
1 deleted
6 unchanged
```

Deltas can now be created in release 2 (deltas 2.2, 2.3, etc.), or another new release can be created in a similar manner.

The help Command

If the command:

```
get lang
```

is now executed, the following message will be output:

```
ERROR [lang]: not an SCCS file (co1)
```

The code **co1** can be used with **help** to print a fuller explanation of the message:

```
help co1
```

This gives the following explanation of why **get lang** produced an error message:

```
co1:
"not an SCCS file"
A file that you think is an SCCS file
does not begin with the characters "s.".
```

help is useful whenever there is doubt about the meaning of almost any SCCS message.

Delta Numbering

Think of deltas as the nodes of a tree in which the root node is the original version of the file. The root is normally named 1.1 and deltas (nodes) are named 1.2, 1.3, etc. The components of these SIDs are called release and level numbers, respectively. Thus, normal naming of new deltas proceeds by incrementing the level number. This is done automatically by SCCS whenever a delta is made.

Because the user may change the release number to indicate a major change, the release number then applies to all new deltas unless specifically changed again. Thus, the evolution of a particular file could be represented by Figure 14-1.

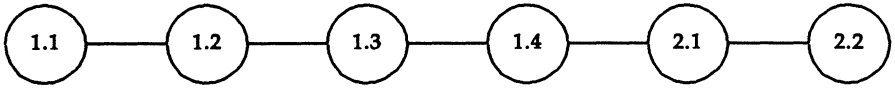

Figure 14-1. Evolution of an SCCS File

This is the normal sequential development of an SCCS file, with each delta dependent on the preceding deltas. Such a structure is called the trunk of an SCCS tree.

There are situations that require branching an SCCS tree. That is, changes are planned to a given delta that will not be dependent on all previous deltas. For example, consider a program in production use at version 1.3 and for which development work on release 2 is already in progress. Release 2 may already have a delta in progress as shown in Figure 14-1. Assume that a production user reports a problem in version 1.3 that cannot wait to be repaired in release 2. The changes necessary to repair the trouble will be applied as a delta to version 1.3 (the version in production use). This creates a new version that will then be released to the user but will not affect the changes being applied for release 2 (i.e., deltas 1.4, 2.1, 2.2, etc.). This new delta is the first node of a new branch of the tree.

Branch delta names always have four SID components: the same release number and level number as the trunk delta, plus a branch number and sequence number. The format is as follows:

release.level.branch.sequence

The branch number of the first delta branching off any trunk delta is always 1, and its sequence number is also 1. For example, the full SID for a delta branching off trunk delta 1.3 will be 1.3.1.1. As other deltas on that same branch are created, only the sequence number changes: 1.3.1.2, 1.3.1.3, etc. This is shown in Figure 14-2.

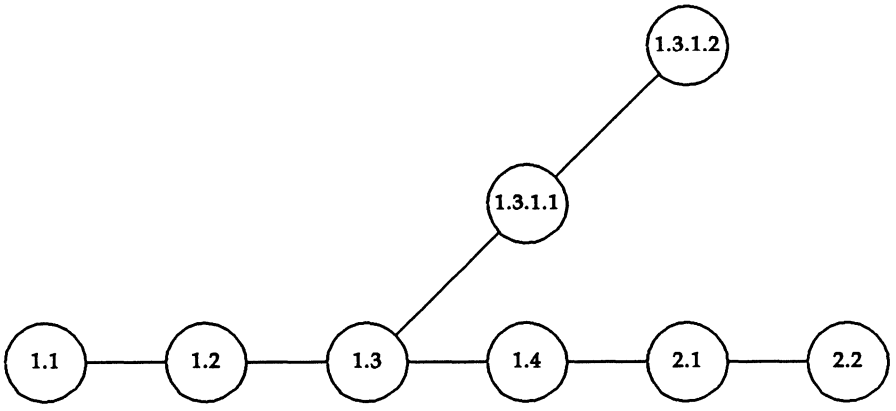

Figure 14-2. Tree Structure with Branch Deltas

The branch number is incremented only when a delta is created that starts a new branch off an existing branch, as shown in Figure 14-3. As this secondary branch develops, the sequence numbers of its deltas are incremented (1.3.2.1, 1.3.2.2, etc.), but the secondary branch number remains the same.

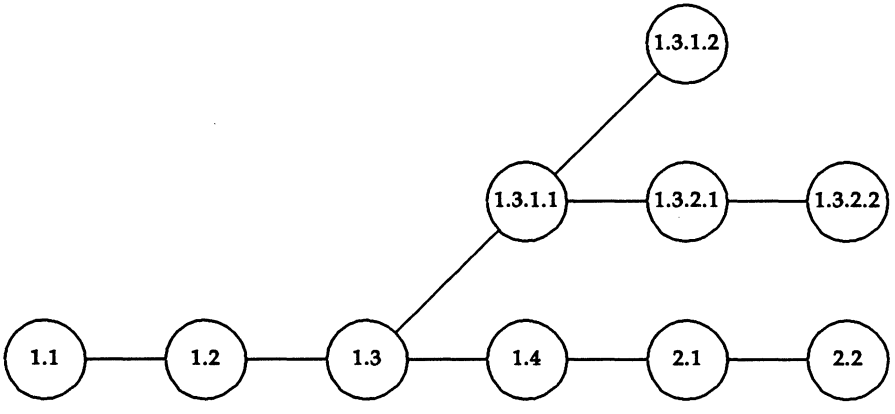

Figure 14-3. Extended Branching Concept

The concept of branching may be extended to any delta in the tree, and the numbering of the resulting deltas proceeds as shown above. SCCS allows the generation of complex tree structures. Although this capability has been provided

for certain specialized uses, the SCCS tree should be kept as simple as possible. Comprehension of its structure becomes difficult as the tree becomes complex.

SCCS Command Conventions

SCCS commands accept two types of arguments:

- keyletters
- filenames

Keyletters are options that begin with a minus sign, **–**, followed by a lowercase letter and, in some cases, a value.

File and/or directory names specify the file(s) the command is to process. Naming a directory is equivalent to naming all the SCCS files within the directory. Non-SCCS files and unreadable files (because of permission modes via **chmod**(1)) in the named directories are silently ignored.

In general, filename arguments may not begin with a minus sign. If a filename of **–** (a lone minus sign) is specified, the command will read the standard input (usually your terminal) for lines and take each line as the name of an SCCS file to be processed. The standard input is read until end-of-file. This feature is often used in pipelines with, for example, the commands **find**(1) or **ls**(1).

Keyletters are processed before filenames. Therefore, the placement of keyletters is arbitrary—that is, they may be interspersed with filenames. Filenames, however, are processed left to right. Somewhat different conventions apply to **help**(1), **what**(1), **sccsdiff**(1), and **val**(1), detailed later under "SCCS Commands."

Certain actions of various SCCS commands are controlled by flags appearing in SCCS files. Some of these flags will be discussed, but for a complete description see **admin**(1) in the *Programmer's Reference Manual.*

The distinction between real user (see **passwd**(1)) and effective user will be of concern in discussing various actions of SCCS commands. For now, assume that the real and effective users are the same—the person logged into the operating system.

x.files and z.files

All SCCS commands that modify an SCCS file do so by writing a copy called the "x.file." This is done to ensure that the SCCS file is not damaged if processing terminates abnormally. SCCS names the x.file by replacing the **s.** of the SCCS filename with **x.**. The x.file is created in the same directory as the SCCS file,

given the same mode (see **chmod**(1)), and is owned by the effective user. When processing is complete, the old SCCS file is destroyed and the modified x.file is renamed (**x.** is replaced by **s.**) and becomes the new SCCS file.

To prevent simultaneous updates to an SCCS file, the same modifying commands also create a lock-file called the "z.file." SCCS forms its name by replacing the **s.** of the SCCS filename with a **z.** prefix. The z.file contains the process number of the command that creates it, and its existence prevents other commands from processing the SCCS file. The z.file is created with access permission mode 444 (read only) in the same directory as the SCCS file and is owned by the effective user. It exists only for the duration of the execution of the command that creates it.

In general, users can ignore x.files and z.files. They are useful only in the event of system crashes or similar situations.

Error Messages

SCCS commands produce error messages on the diagnostic output in this format:

```
ERROR [name-of-file-being-processed]: message text (code)
```

The code in parentheses can be used as an argument to the **help** command to obtain a further explanation of the message. Detection of a fatal error during the processing of a file causes the SCCS command to stop processing that file and proceed with the next file specified.

SCCS Commands

This section describes the major features of the fourteen SCCS commands and their most common arguments. Full descriptions with details of all arguments are in the *Programmer's Reference Manual*.

Here is a quick-reference overview of the commands:

get	retrieves versions of SCCS files
unget	undoes the effect of a **get –e** prior to the file being **delta**ed
delta	applies deltas (changes) to SCCS files and creates new versions
admin	initializes SCCS files, manipulates their descriptive text, and controls delta creation rights
prs	prints portions of an SCCS file in user specified format

sact	prints information about files that are currently out for edit
help	gives explanations of error messages
rmdel	removes a delta from an SCCS file allows removal of deltas created by mistake
cdc	changes the commentary associated with a delta
what	searches any operating system file(s) for all occurrences of a special pattern and prints out what follows it useful in finding identifying information inserted by the **get** command
sccsdiff	shows differences between any two versions of an SCCS file
comb	combines consecutive deltas into one to reduce the size of an SCCS file
val	validates an SCCS file
vc	a filter that may be used for version control

The get Command

The **get**(1) command creates a file that contains a specified version of an SCCS file. The version is retrieved by beginning with the initial version and then applying deltas, in order, until the desired version is obtained. The resulting file is called the "g.file." It is created in the current directory and is owned by the real user. The mode assigned to the g.file depends on how the **get** command is used.

The most common use of **get** is:

```
get s.abc
```

which normally retrieves the latest version of file **abc** from the SCCS file tree trunk and produces (for example) on the standard output:

```
1.3
67 lines
No id keywords (cm7)
```

meaning version 1.3 of file **s.abc** was retrieved (assuming 1.3 is the latest trunk delta), it has 67 lines of text, and no ID keywords were substituted in the file.

The generated g.file (file **abc**) is given access permission mode 444 (read only). This particular way of using **get** is intended to produce g.files only for inspection, compilation, etc. It is not intended for editing (making deltas).

When several files are specified, the same information is output for each one. For example, the command:

```
get s.abc s.xyz
```

produces:

```
s.abc:
1.3
67 lines
No id keywords (cm7)

s.xyz:
1.7
85 lines
No id keywords (cm7)
```

ID Keywords

In generating a g.file for compilation, it is useful to record the date and time of creation, the version retrieved, the module's name, etc. within the g.file. This information appears in a load module when one is eventually created. SCCS provides a convenient mechanism for doing this automatically. Identification (ID) keywords appearing anywhere in the generated file are replaced by appropriate values according to the definitions of those ID keywords. The format of an ID keyword is an uppercase letter enclosed by percent signs, %. For example, the ID keyword replaced by the SID of the retrieved version of a file is:

```
%I%
```

Similarly, **%H%** and **%M%** are the names of the g.file. Thus, executing **get** on an SCCS file that contains the PL/I declaration,

```
DCL ID CHAR(100) VAR INIT('%M% %I% %H%');
```

gives (for example) the following:

```
DCL ID CHAR(100) VAR INIT('MODNAME 2.3 07/18/85');
```

When no ID keywords are substituted by **get**, the following message is issued:

```
No id keywords (cm7)
```

This message is normally treated as a warning by **get** although the presence of the **i** flag in the SCCS file causes it to be treated as an error. For a complete list of the approximately twenty ID keywords provided, see **get**(1) in the *Programmer's Reference Manual.*

Retrieval of Different Versions

The version of an SCCS file that **get** retrieves is the most recently created delta of the highest numbered trunk release. However, any other version can be retrieved with **get –r** by specifying the version's SID. Thus, the command:

```
get –r1.3 s.abc
```

retrieves version 1.3 of file **s.abc** and produces (for example) on the standard output:

```
1.3
64 lines
```

A branch delta may be retrieved similarly, with a command such as:

```
get –r1.5.2.3 s.abc
```

which produces (for example) on the standard output:

```
1.5.2.3
234 lines
```

When a SID is specified and the particular version does not exist in the SCCS file, an error message results.

Omitting the level number, as in:

```
get –r3 s.abc
```

causes retrieval of the trunk delta with the highest level number within the given release. Thus, the above command might output:

```
3.7
213 lines
```

If the given release does not exist, **get** retrieves the trunk delta with the highest level number within the highest-numbered existing release that is lower than the given release. For example, assume release 9 does not exist in file **s.abc** and release 7 is the highest-numbered release below 9. Executing the command:

```
get –r9 s.abc
```

might produce:

```
7.6
420 lines
```

which indicates that trunk delta 7.6 is the latest version of file **s.abc** below release 9. Similarly, omitting the sequence number, as in:

```
get -r4.3.2 s.abc
```

results in the retrieval of the branch delta with the highest sequence number on the given branch. (If the given branch does not exist, an error message results.) This might result in the following output:

```
4.3.2.8
89 lines
```

get -t will retrieve the latest (top) version of a particular release when no **-r** is used or when its value is simply a release number. The latest version is the delta produced most recently, independent of its location on the SCCS file tree. Thus, if the most recent delta in release 3 is 3.5, the command:

```
get -r3 -t s.abc
```

might produce:

```
3.5
59 lines
```

However, if branch delta 3.2.1.5 were the latest delta (created after delta 3.5), the same command might produce:

```
3.2.1.5
46 lines
```

Retrieval With Intent to Make a Delta

get -e indicates an intent to make a delta. First, **get** checks the following items:

1. The user list to determine whether the login name or group ID of the person executing **get** is present. The login name or group ID must be present for the user to be allowed to make deltas. (See "The **admin** Command" for a discussion of making user lists.)

2. Whether the release number (R) of the version being retrieved satisfies the relation:

   floor is less than or equal to R, which is
   less than or equal to ceiling

   to determine if the release being accessed is a protected release. The floor and ceiling are flags in the SCCS file representing start and end of range.

3. That the R is not locked against editing. The lock is a flag in the SCCS file.

4. Whether multiple concurrent edits are allowed for the SCCS file by the **j** flag in the SCCS file.

A failure of any of the first three conditions causes the processing of the corresponding SCCS file to terminate.

If the above checks succeed, **get –e** causes the creation of a g.file in the current directory with mode 644 (readable by everyone, writable only by the owner) owned by the real user. If a writable g.file already exists, **get** terminates with an error. This is to prevent inadvertent destruction of a g.file being edited to make a delta.

Any ID keywords appearing in the g.file are not substituted by **get –e** because the generated g.file is subsequently used to create another delta. Replacement of ID keywords causes them to be permanently changed in the SCCS file. Because of this, **get** does not need to check for their presence in the g.file. Thus, the message:

```
No id keywords (cm7)
```

is never output when **get –e** is used.

In addition, **get –e** causes the creation (or updating) of a p.file that is used to pass information to the **delta** command.

The command:

```
get –e s.abc
```

produces (for example) on the standard output:

```
1.3
new delta 1.4
67 lines
```

Undoing a get –e

There may be times when a file is retrieved for editing in error; there is really no editing that needs to be done at this time. In such cases, the **unget** command can be used to cancel the delta reservation that was set up.

Additional get Options

If **get –r** and/or **–t** are used together with **–e**, the version retrieved for editing is the one specified with **–r** and/or **–t**.

get –i and **–x** are used to specify a list (see **get**(1) in the *Programmer's Reference Manual* for the syntax of such a list) of deltas to be included and excluded,

respectively. Including a delta means forcing its changes to be included in the retrieved version. This is useful in applying the same changes to more than one version of the SCCS file. Excluding a delta means forcing it not to be applied. This may be used to undo the effects of a previous delta in the version to be created.

Whenever deltas are included or excluded, **get** checks for possible interference with other deltas. Two deltas can interfere, for example, when each one changes the same line of the retrieved g.file. A warning shows the range of lines within the retrieved g.file where the problem may exist. The user should examine the g.file to determine what the problem is and take corrective steps (e.g., edit the file).

> **CAUTION**
>
> **get –i** and **get –x** should be used with extreme care.

get –k is used either to regenerate a g.file that may have been accidentally removed or ruined after **get –e**, or simply to generate a g.file in which the replacement of ID keywords has been suppressed. A g.file generated by **get –k** is identical to one produced by **get –e**, but no processing related to the p.file takes place.

Concurrent Edits of Different SID

The ability to retrieve different versions of an SCCS file allows several deltas to be in progress at any given time. This means that several **get –e** commands may be executed on the same file as long as no two executions retrieve the same version (unless multiple concurrent edits are allowed).

The p.file created by **get –e** is named by automatic replacement of the SCCS filename's prefix **s.** with **p.**. It is created in the same directory as the SCCS file, given mode 644 (readable by everyone, writable only by the owner), and owned by the effective user. The p.file contains the following information for each delta that is still in progress:

- the SID of the retrieved version
- the SID given to the new delta when it is created
- the login name of the real user executing **get**

The first execution of **get –e** causes the creation of a p.file for the corresponding SCCS file. Subsequent executions only update the p.file with a line containing the above information. Before updating, however, **get** checks to assure that no entry already in the p.file specifies that the SID of the version to be retrieved is already retrieved (unless multiple concurrent edits are allowed). If the check succeeds, the user is informed that other deltas are in progress and processing continues. If the check fails, an error message results.

Note that concurrent executions of **get** must be carried out from different directories. Subsequent executions from the same directory will attempt to overwrite the g.file, which is an SCCS error condition. In practice, this problem does not arise since each user normally has a different working directory. See "Protection" under "SCCS Files" for a discussion of how different users are permitted to use SCCS commands on the same files.

Figure 14-4 shows the possible SID components a user can specify with **get** (left-most column), the version that will then be retrieved by **get**, and the resulting SID for the delta, which **delta** will create (right-most column).

SID Specified in get*	–b Key-Letter Used†	Other Conditions	SID Retrieved by get	SID of Delta To be Created by delta
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1)
R	no	R > mR	mR.mL	R.1§
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1

Figure 14-4. Determination of New SID (sheet 1 of 2)

SID Specified in get*	−b Key-Letter Used†	Other Conditions	SID Retrieved by get	SID of Delta To be Created by delta
R	–	R< mR and R does not exist	hR.mL**	hR.mL.(mB+1).1
R	–	Trunk successor number in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L.	no	No trunk successor	R.L	R.(L+1)
R.L.	yes	No trunk successor	R.L	R.L.(mB+1).1
R.L	–	Trunk successor in release ≥ R	R.L	R.L.(mS+1).1
R.L.B	no	No branch successor	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch successor	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch successor	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch successor	R.L.B.S	R.L.(mB+1).1
R.L.B.S	–	Branch successor	R.L.B.S	R.L.(mB+1).1

Figure 14-4. Determination of New SID (sheet 2 of 2)

Footnotes to Figure 14-4:

* R, L, B, and S mean release, level, branch, and sequence numbers in the SID, and m means maximum. Thus, for example, R.mL means the maximum level number within release R. R.L.(mB+1).1 means the first

sequence number on the new branch (i.e., maximum branch number plus 1) of level L within release R. Note that if the SID specified is R.L, R.L.B, or R.L.B.S, each of these specified SID numbers must exist.

† The **–b** keyletter is effective only if the **b** flag (see **admin(1)**) is present in the file. An entry of – means irrelevant.

‡ This case applies if the **d** (default SID) flag is not present. If the **d** flag is present in the file, the SID is interpreted as if specified on the command line. Thus, one of the other cases in this figure applies.

§ This is used to force the creation of the first delta in a new release.

** hR is the highest existing release that is lower than the specified, nonexistent release R.

Concurrent Edits of Same SID

Under normal conditions, more than one **get –e** for the same SID is not permitted. That is, **delta** must be executed before a subsequent **get –e** is executed on the same SID.

Multiple concurrent edits are allowed if the **j** flag is set in the SCCS file. Thus, the command and output:

```
get –e s.abc
1.1
new delta 1.2
5 lines
```

may be immediately followed by:

```
get –e s.abc
1.1
new delta 1.1.1.1
5 lines
```

without an intervening **delta**. In this case, a **delta** after the first **get** will produce delta 1.2 (assuming 1.1 is the most recent trunk delta), and a **delta** after the second **get** will produce delta 1.1.1.1.

Keyletters That Affect Output

get –p causes the retrieved text to be written to the standard output rather than to a g.file. In addition, all output normally directed to the standard output (such as

the SID of the version retrieved and the number of lines retrieved) is directed instead to the diagnostic output. **get –p** is used, for example, to create a g.file with an arbitrary name, as in:

get –p s.abc > *arbitrary-file-name*

get –s suppresses output normally directed to the standard output, such as the SID of the retrieved version and the number of lines retrieved, but it does not affect messages normally directed to the diagnostic output. **get –s** is used to prevent nondiagnostic messages from appearing on the user's terminal and is often used with **–p** to pipe the output, as in:

get –p –s s.abc | pg

get –g suppresses the retrieval of the text of an SCCS file. This is useful in several ways. For example, to verify a particular SID in an SCCS file, the command:

get –g –r4.3 s.abc

outputs the SID 4.3 if it exists in the SCCS file **s.abc** or an error message if it does not. Another use of **get –g** is in regenerating a p.file that may have been accidentally destroyed, as in:

get –e –g s.abc

get –l causes SCCS to create an "l.file." It is named by replacing the **s.** of the SCCS filename with **l.**, created in the current directory with mode 444 (read only) and owned by the real user. The l.file contains a table (whose format is described under **get**(1) in the *Programmer's Reference Manual*) showing the deltas used in constructing a particular version of the SCCS file. For example, the command:

get –r2.3 –l s.abc

generates an l.file showing the deltas applied to retrieve version 2.3 of file **s.abc**. Specifying **p** with **–l**, as in:

get –lp –r2.3 s.abc

causes the output to be written to the standard output rather than to the l.file. **get –g** can be used with **–l** to suppress the retrieval of the text.

get –m identifies the changes applied to an SCCS file. Each line of the g.file is preceded by the SID of the delta that caused the line to be inserted. The SID is separated from the text of the line by a tab character.

get –n causes each line of a g.file to be preceded by the value of the ID keyword

and a tab character. This is most often used in a pipeline with **grep**(1). For example, to find all lines that match a given pattern in the latest version of each SCCS file in a directory, the following may be executed:

get –p –n –s *directory* **|** **grep** *pattern*

If both **–m** and **–n** are specified, each line of the generated g.file is preceded by the value of the **chap3.13** ID keyword and a tab (this is the effect of **–n**) and is followed by the line in the format produced by **–m**. Because use of **–m** and/or **–n** causes the contents of the g.file to be modified, such a g.file must not be used for creating a delta. Therefore, neither **–m** nor **–n** may be specified together with **get –e**.

NOTE

See **get**(1) in the *Programmer's Reference Manual* for a full description of additional keyletters.

The delta Command

The **delta**(1) command is used to incorporate changes made to a g.file into the corresponding SCCS file—that is, to create a delta and, therefore, a new version of the file.

The **delta** command requires the existence of a p.file (created via **get –e**). It examines the p.file to verify the presence of an entry containing the user's login name. If none is found, an error message results.

get –e performs. If all checks are successful, **delta** determines what has been changed in the g.file by comparing it via **diff**(1) with its own temporary copy of the g.file as it was before editing. This temporary copy of the g.file is called the d.file and is obtained by performing an internal **get** on the SID specified in the p.file entry.

The required p.file entry is the one containing the login name of the user executing **delta**, because the user who retrieved the g.file must be the one who creates the delta. However, if the login name of the user appears in more than one entry, the same user has executed **get –e** more than once on the same SCCS file. Then, **delta –r** must be used to specify the SID that uniquely identifies the p.file entry. This entry is then the one used to obtain the SID of the delta to be created.

In practice, the most common use of **delta** is:

delta s.abc

which prompts:

```
comments?
```

to which the user replies with a description of why the delta is being made, ending the reply with a newline character. The user's response may be up to 512 characters long with newlines (not intended to terminate the response) escaped by backslashes, \.

If the SCCS file has a **v** flag, **delta** first prompts with:

```
MRs?
```

(Modification Requests), on the standard output. The standard input is then read for MR numbers, separated by blanks and/or tabs, ended with a newline character. A Modification Request is a formal way of asking for a correction or enhancement to the file. In some controlled environments where changes to source files are tracked, deltas are permitted only when initiated by a trouble report, change request, trouble ticket, etc., collectively called MRs. Recording MR numbers within deltas is a way of enforcing the rules of the change management process.

delta –y and/or **–m** can be used to enter comments and MR numbers on the command line rather than through the standard input, as in:

delta –y"*descriptive comment*" **–m**"*mrnum1 mrnum2*" **s.abc**

In this case, the prompts for comments and MRs are not printed, and the standard input is not read. These two keyletters are useful when **delta** is executed from within a shell procedure (see **sh**(1) in the *Programmer's Reference Manual*).

NOTE

delta –m is allowed only if the SCCS file has a **v** flag.

No matter how comments and MR numbers are entered with **delta**, they are recorded as part of the entry for the delta being created. Also, they apply to all SCCS files specified with the **delta**.

If **delta** is used with more than one file argument and the first file named has a **v** flag, all files named must have this flag. Similarly, if the first file named does not have the flag, none of the files named may have it.

When **delta** processing is complete, the standard output displays the SID of the new delta (from the p.file) and the number of lines inserted, deleted, and left unchanged. For example:

```
1.4
14 inserted
7 deleted
345 unchanged
```

If line counts do not agree with the user's perception of the changes made to a g.file, it may be because there are various ways to describe a set of changes, especially if lines are moved around in the g.file. However, the total number of lines of the new delta (the number inserted plus the number left unchanged) should always agree with the number of lines in the edited g.file.

If you are in the process of making a delta, the **delta** command finds no ID keywords in the edited g.file, the message:

```
No id keywords (cm7)
```

is issued after the prompts for commentary but before any other output. This means that any ID keywords that may have existed in the SCCS file have been replaced by their values or deleted during the editing process. This could be caused by making a delta from a g.file that was created by a **get** without **–e** (ID keywords are replaced by **get** in such a case). It could also be caused by accidentally deleting or changing ID keywords while editing the g.file. Or, it is possible that the file had no ID keywords. In any case, the delta will be created unless there is an **i** flag in the SCCS file (meaning the error should be treated as fatal), in which case the delta will not be created.

After the processing of an SCCS file is complete, the corresponding p.file entry is removed from the p.file. All updates to the p.file are made to a temporary copy, the "q.file," whose use is similar to the use of the x.file described earlier under "SCCS Command Conventions." If there is only one entry in the p.file, then the p.file itself is removed.

In addition, **delta** removes the edited g.file unless **–n** is specified. For example, the command:

```
delta –n s.abc
```

will keep the g.file after processing.

delta –s suppresses all output normally directed to the standard output, other than `comments?` and `MRs?`. Thus, use of **–s** with **–y** (and/or **–m**) causes **delta** to neither read the standard input nor write the standard output.

The differences between the g.file and the d.file constitute the delta and may be printed on the standard output by using **delta –p**. The format of this output is similar to that produced by **diff**(1).

The admin Command

The **admin**(1) command is used to administer SCCS files—that is, to create new SCCS files and change the parameters of existing ones. When an SCCS file is created, its parameters are initialized by use of keyletters with **admin** or are assigned default values if no keyletters are supplied. The same keyletters are used to change the parameters of existing SCCS files.

Two keyletters are used in detecting and correcting corrupted SCCS files (see "Auditing" under "SCCS Files").

Newly created SCCS files are given access permission mode 444 (read only) and are owned by the effective user. Only a user with write permission in the directory containing the SCCS file may use the **admin** command on that file.

Creation of SCCS Files

An SCCS file can be created by executing the command:

admin –ifirst s.abc

in which the value **first** with **–i** is the name of a file from which the text of the initial delta of the SCCS file **s.abc** is to be taken. Omission of a value with **–i** means **admin** is to read the standard input for the text of the initial delta.

The command:

admin –i s.abc < first

is equivalent to the previous example.

If the text of the initial delta does not contain ID keywords, the message:

```
No id keywords (cm7)
```

is issued by **admin** as a warning. However, if the command also sets the **i** flag (not to be confused with the **–i** keyletter), the message is treated as an error and the SCCS file is not created. Only one SCCS file may be created at a time using **admin –i**.

admin –r is used to specify a release number for the first delta. Thus, the command:

admin –ifirst –r3 s.abc

means the first delta should be named 3.1 rather than the normal 1.1. Because **–r** has meaning only when creating the first delta, its use is permitted only with **–i**.

Inserting Commentary for the Initial Delta

When an SCCS file is created, the user may want to record why this was done. Comments (**admin –y**) and/or MR numbers (**–m**) can be entered in exactly the same way as a **delta**.

If **–y** is omitted, a comment line of the form:

date and time created YY/MM/DD HH:MM:SS by logname

is automatically generated.

If it is desired to supply MR numbers (**admin –m**), the **v** flag must be set via **–f**. The **v** flag simply determines whether MR numbers must be supplied when using any SCCS command that modifies a delta commentary (see **sccsfile**(4) in the *Programmer's Reference Manual*) in the SCCS file. Thus:

admin –ifirst –m*mrnum1* **–fv s.abc**

Note that **–y** and **–m** are effective only if a new SCCS file is being created.

Initialization and Modification of SCCS File Parameters

Part of an SCCS file is reserved for descriptive text, usually a summary of the file's contents and purpose. It can be initialized or changed by using **admin –t**.

When an SCCS file is first being created and **–t** is used, it must be followed by the name of a file from which the descriptive text is to be taken. For example, the command:

admin –ifirst –tdesc s.abc

specifies that the descriptive text is to be taken from file **desc**.

When processing an existing SCCS file, **–t** specifies that the descriptive text (if any) currently in the file is to be replaced with the text in the named file. Thus, the command:

admin –tdesc s.abc

specifies that the descriptive text of the SCCS file is to be replaced by the contents of **desc**.

Omission of the filename after the **–t** keyletter, as in:

admin –t s.abc

causes the removal of the descriptive text from the SCCS file.

The flags of an SCCS file may be initialized or changed by **admin –f**, or deleted via **–d**.

SCCS file flags are used to direct certain actions of the various commands. (See **admin**(1) in the *Programmer's Reference Manual* for a description of all the flags.) For example, the **i** flag specifies that a warning message (stating that there are no ID keywords contained in the SCCS file) should be treated as an error. The **d** (default SID) flag specifies the default version of the SCCS file to be retrieved by the **get** command.

admin –f is used to set flags and, if desired, their values. For example, the command:

admin –ifirst –fi –fm*modname* **s.abc**

sets the **i** and **m** (module name) flags. The value *modname* specified for the **m** flag is the value that the **get** command will use to replace the **%M%** ID keyword. (In the absence of the **m** flag, the name of the g.file is used as the replacement for the **%M%** ID keyword.) Several **–f** keyletters may be supplied on a single **admin**, and they may be used whether the command is creating a new SCCS file or processing an existing one.

admin –d is used to delete a flag from an existing SCCS file. As an example, the command:

admin –dm s.abc

removes the **m** flag from the SCCS file. Several **–d** keyletters may be used with one **admin** and may be intermixed with **–f**.

SCCS files contain a list of login names and/or group IDs of users who are allowed to create deltas. This list is empty by default, allowing anyone to create deltas. To create a user list (or add to an existing one), **admin –a** is used. For example, the command:

admin –axyz –awql –a1234 s.abc

adds the login names **xyz** and **wql** and the group ID **1234** to the list. **admin –a** may be used whether creating a new SCCS file or processing an existing one.

admin –e (erase) is used to remove login names or group IDs from the list.

The prs Command

The **prs**(1) command is used to print all or part of an SCCS file on the standard output. If **prs –d** is used, the output will be in a format called data specification. Data specification is a string of SCCS file data keywords (not to be confused with **get** ID keywords) interspersed with optional user text.

Data keywords are replaced by appropriate values according to their definitions. For example, the symbol:

:I:

is defined as the data keyword replaced by the SID of a specified delta. Similarly, **:F:** is the data keyword for the SCCS filename currently being processed, and **:C:** is the comment line associated with a specified delta. All parts of an SCCS file have an associated data keyword. For a complete list, see **prs**(1) in the *Programmer's Reference Manual.*

There is no limit to the number of times a data keyword may appear in a data specification. Thus, for example, the command line:

prs –d":I: this is the top delta for :F: :I:" s.abc

may produce on the standard output:

```
2.1 this is the top delta for s.abc 2.1
```

Information may be obtained from a single delta by specifying its SID using **prs –r**. For example, the command line:

prs –d":F:: :I: comment line is: :C:" –r1.4 s.abc

may produce the following output:

s.abc: 1.4 comment line is: THIS IS A COMMENT

If **–r** is not specified, the value of the SID defaults to the most recently created delta.

In addition, information from a range of deltas may be obtained with **–l** or **–e**. The use of **prs –e** substitutes data keywords for the SID designated via **–r** and all deltas created earlier, while **prs –l** substitutes data keywords for the SID designated via **–r** and all deltas created later. Thus, the command:

prs –d:I: –r1.4 –e s.abc

may output:

```
1.4
1.3
1.2.1.1
1.2
1.1
```

and the command:

```
prs -d:I: -r1.4 -l s.abc
```

may produce:

```
3.3
3.2
3.1
2.2.1.1
2.2
2.1
1.4
```

Substitution of data keywords for all deltas of the SCCS file may be obtained by specifying both **-e** and **-l**.

The sact Command

sact(1) is like a special form of the **prs** command that produces a report about files that are out for edit. The command takes only one type of argument: a list of file or directory names. The report shows the SID of any file in the list that is out for edit, the SID of the impending delta, the login of the user who executed the **get -e** command, and the date and time the **get -e** was executed. It is a useful command for an administrator.

The help Command

The **help**(1) command prints the syntax of SCCS commands and of messages that may appear on the user's terminal. Arguments to **help** are simply SCCS commands or the code numbers that appear in parentheses after SCCS messages. (If no argument is given, **help** prompts for one.) Explanatory information is printed on the standard output. If no information is found, an error message is printed. When more than one argument is used, each is processed independently, and an error resulting from one will not stop the processing of the others.

NOTE

There is no conflict between the **help**(1) command of SCCS and the operating system **help**(1) utilities. The installation procedure for each package checks for the prior existence of the other.

Explanatory information related to a command is a synopsis of the command. For example, the command:

```
help ge5 rmdel
```

produces:

```
ge5:
"nonexistent sid"
The specified sid does not exist in the
given file.
Check for typos.

rmdel:
  rmdel  -rSID  name  ...
```

The rmdel Command

The **rmdel**(1) command allows removal of a delta from an SCCS file. Its use should be reserved for deltas in which incorrect global changes were made. The delta to be removed must be a leaf delta. That is, it must be the most recently created delta on its branch or on the trunk of the SCCS file tree. In Figure 14-3, only deltas 1.3.1.2, 1.3.2.2, and 2.2 can be removed. Only after they are removed can deltas 1.3.2.1 and 2.1 be removed.

To be allowed to remove a delta, the effective user must have write permission in the directory containing the SCCS file. In addition, the real user must be either the one who created the delta being removed or the owner of the SCCS file and its directory.

The **–r** keyletter is mandatory with **rmdel**. It is used to specify the complete SID of the delta to be removed. Thus, the command:

```
rmdel –r2.3 s.abc
```

specifies the removal of trunk delta 2.3.

Before removing the delta, **rmdel** checks that the release number (R) of the given SID satisfies the relation:

floor less than or equal to R less than or equal to ceiling

The **rmdel** command also checks the SID to make sure it is not for a version on which a **get** for editing has been executed and whose associated **delta** has not yet been made. In addition, the login name or group ID of the user must appear in the file's user list (or the user list must be empty). Also, the release specified cannot be locked against editing. That is, if the **l** flag is set (see **admin**(1) in the *Programmer's Reference Manual*), the release must not be contained in the list. If these conditions are not satisfied, processing is terminated, and the delta is not removed.

Once a specified delta has been removed, its type indicator in the delta table of the SCCS file is changed from D (delta) to R (removed).

The cdc Command

The **cdc**(1) command is used to change the commentary made when the delta was created. It is similar to the **rmdel** command (e.g., **–r** and full SID are necessary), although the delta need not be a leaf delta. For example, the command:

```
cdc –r3.4 s.abc
```

specifies that the commentary of delta 3.4 is to be changed. New commentary is then prompted for as with **delta**.

The old commentary is kept, but it is preceded by a comment line indicating that it has been superseded, and the new commentary is entered ahead of the comment line. The inserted comment line records the login name of the user executing **cdc** and the time of its execution.

The **cdc** command also allows for the insertion of new and deletion of old ("!" prefix) MR numbers. Thus, the command:

```
cdc –r1.4 s.abc
MRs? mrnum3 !mrnum1          (The MRs? prompt appears only
                              if the v flag has been set.)
comments? deleted wrong MR number and inserted correct MR number
```

inserts **mrnum3** and deletes **mrnum1** for delta 1.4.

NOTE

An MR (Modification Request) is described above under the **delta** command.

The what Command

The **what**(1) command is used to find identifying information within any file whose name is given as an argument. No keyletters are accepted. The **what** command searches the given file(s) for all occurrences of the string **@(#)**, which is the replacement for the **%Z%** ID keyword (see **get**(1)). It prints on the standard output whatever follows the string until the first double quote, ", greater than, >, backslash, \, newline, or nonprinting NUL character.

For example, if an SCCS file called **s.prog.c** (a C language program) contains the following line:

```
char  id[]= "%W%";
```

and if the command:

get –r3.4 s.prog.c

is used, the resulting g.file is compiled to produce **prog.o** and **a.out**. Then, the command:

what prog.c prog.o a.out

produces:

```
prog.c:
  prog.c:  3.4
prog.o:
  prog.c:  3.4
a.out:
  prog.c:  3.4
```

The string searched for by **what** need not be inserted via an ID keyword of **get**; it may be inserted in any convenient manner.

The sccsdiff Command

The **sccsdiff**(1) command determines (and prints on the standard output) the differences between any two versions of an SCCS file. The versions to be compared are specified with **sccsdiff –r** in the same way as with **get –r**. SID numbers must be specified as the first two arguments. Any following keyletters

are interpreted as arguments to the **pr**(1) command (which prints the differences) and must appear before any filenames. The SCCS file(s) to be processed are named last. Directory names and a name of **–** (a lone minus sign) are not acceptable to **sccsdiff**.

The following is an example of the format of **sccsdiff**:

```
sccsdiff –r3.4 –r5.6 s.abc
```

The differences are printed the same way as by **diff**(1).

The comb Command

The **comb**(1) command lets the user try to reduce the size of an SCCS file. It generates a shell procedure (see **sh**(1) in the *Programmer's Reference Manual*) on the standard output, which reconstructs the file by discarding unwanted deltas and combining other specified deltas. (It is not recommended that **comb** be used as a matter of routine.)

In the absence of any keyletters, **comb** preserves only leaf deltas and the minimum number of ancestor deltas necessary to preserve the shape of an SCCS tree. The effect of this is to eliminate middle deltas on the trunk and on all branches of the tree. Thus, in Figure 14-3, deltas 1.2, 1.3.2.1, 1.4, and 2.1 would be eliminated.

Some of the keyletters used with this command are:

comb –s This option generates a shell procedure that produces a report of the percentage space (if any) the user will save. This is often useful as an advance step.

comb –p This option is used to specify the oldest delta the user wants preserved.

comb –c This option is used to specify a list (see **get**(1) in the *Programmer's Reference Manual* for its syntax) of deltas the user wants preserved. All other deltas will be discarded.

The shell procedure generated by **comb** is not guaranteed to save space. A reconstructed file may even be larger than the original. Note, too, that the shape of an SCCS file tree may be altered by the reconstruction process.

The val Command

The **val**(1) command is used to determine whether a file is an SCCS file meeting the characteristics specified by certain keyletters. It checks for the existence of a particular delta when the SID for that delta is specified with **–r**.

The string following **–y** or **–m** is used to check the value set by the **t** or **m** flag, respectively. See **admin**(1) in the *Programmer's Reference Manual* for descriptions of these flags.

The **val** command treats the special argument **–** differently from other SCCS commands. It allows **val** to read the argument list from the standard input instead of from the command line, and the standard input is read until an end-of-file (CTRL-D) is entered. This permits one **val** command with different values for keyletters and file arguments. For example, the command:

```
val – –yc –mabc s.abc –mxyz –ypl1 s.xyz
```

first checks if file **s.abc** has a value **c** for its type flag and value **abc** for the module name flag. Once this is done, **val** processes the remaining file, in this case **s.xyz**.

The **val** command returns an 8-bit code. Each bit set shows a specific error (see **val**(1) for a description of errors and codes). In addition, an appropriate diagnostic is printed unless suppressed by **–s**. A return code of 0 means all files met the characteristics specified.

The vc Command

The **vc**(1) command is an **awk**-like tool used for version control of sets of files. While it is distributed as part of the SCCS package, it does not require the files it operates on to be under SCCS control. A complete description of **vc** may be found in the *Programmer's Reference Manual.*

SCCS Files

This section covers protection mechanisms used by SCCS, the format of SCCS files, and the recommended procedures for auditing SCCS files.

Protection

SCCS relies on the capabilities of the operating system for most of the protection mechanisms required to prevent unauthorized changes to SCCS files—that is, changes by non-SCCS commands. Protection features provided directly by SCCS are the release lock flag, the release floor and ceiling flags, and the user list.

Files created by the **admin** command are given access permission mode 444 (read only). This mode should remain unchanged because it prevents modification of SCCS files by non-SCCS commands. Directories containing SCCS files should be given mode 755, which allows only the owner of the directory to modify it.

SCCS files should be kept in directories that contain only SCCS files and any temporary files created by SCCS commands. This simplifies their protection and auditing. The contents of directories should be logical groupings—subsystems of the same large project, for example.

SCCS files should have only one link (name) because commands that modify them do so by creating a copy of the file (the x.file; see "SCCS Command Conventions"). When processing is done, the old file is automatically removed and the x.file renamed (**s.** prefix). If the old file had additional links, this breaks them. Then, rather than process such files, SCCS commands will produce an error message.

When only one person uses SCCS, the real and effective user IDs are the same; and the user ID owns the directories containing SCCS files. Therefore, SCCS may be used directly without any preliminary preparation.

When several users with unique user IDs are assigned SCCS responsibilities (e.g., on large development projects), one user—that is, one user ID—must be chosen as the owner of the SCCS files. This person will administer the files (e.g. use the **admin** command) and will be SCCS administrator for the project. Because other users do not have the same privileges and permissions as the SCCS administrator, they are not able to execute directly those commands that require write permission in the directory containing the SCCS files. Therefore, a project-dependent program is required to provide an interface to the **get**, **delta**, and, if desired, **rmdel** and **cdc** commands.

The interface program must be owned by the SCCS administrator and must have the set user ID on execution bit on (see **chmod**(1) in the *User's Reference Manual*). This assures that the effective user ID is the user ID of the SCCS administrator. With the privileges of the interface program during command execution, the owner of an SCCS file can modify it at will. Other users whose login names or group IDs are in the user list for that file (but are not the owner) are given the

necessary permissions only for the duration of the execution of the interface program. Thus, they may modify SCCS only with **delta** and, possibly, **rmdel** and **cdc**.

A project-dependent interface program, as its name implies, can be custom built for each project. Its creation is discussed later under "An SCCS Interface Program."

Formatting

SCCS files are composed of lines of ASCII text arranged in six parts as follows:

Checksum	a line containing the logical sum of all the characters of the file (not including the checksum itself)
Delta Table	information about each delta, such as type, SID, date and time of creation, and commentary
User Names	list of login names and/or group IDs of users who are allowed to modify the file by adding or removing deltas
Flags	indicators that control certain actions of SCCS commands
Descriptive Text	usually a summary of the contents and purpose of the file
Body	the text administered by SCCS, intermixed with internal SCCS control lines

Details on these file sections may be found in **sccsfile**(4). The checksum is discussed below under "Auditing."

Since SCCS files are ASCII files they can be processed by non-SCCS commands like **ed**(1), **grep**(1), and **cat**(1). This is convenient when an SCCS file must be modified manually (e.g., a delta's time and date were recorded incorrectly because the system clock was set incorrectly), or when a user wants simply to look at the file.

CAUTION

Extreme care should be exercised when modifying SCCS files with non-SCCS commands.

Auditing

When a system or hardware malfunction destroys an SCCS file, any command will issue an error message. Commands also use the checksum stored in an SCCS file to determine whether the file has been corrupted since it was last accessed (possibly by having lost one or more blocks or by having been modified with **ed**(1)). No SCCS command will process a corrupted SCCS file except the **admin** command with **–h** or **–z**, as described below.

SCCS files should be audited for possible corruptions on a regular basis. The simplest and fastest way to do an audit is to use **admin –h** and specify all SCCS files, as in:

admin –h s.*file1* **s.***file2* ...

or

admin –h *directory1 directory2* ...

If the new checksum of any file is not equal to the checksum in the first line of that file, the message:

```
corrupted file (co6)
```

is produced for that file. The process continues until all specified files have been examined. When examining directories (as in the second example above), the checksum process will not detect missing files. A simple way to learn whether files are missing from a directory is to execute the **ls**(1) command periodically, and compare the outputs. Any file whose name appeared in a previous output but not in the current one no longer exists.

When a file has been corrupted, the way to restore it depends on the extent of the corruption. If damage is extensive, the best solution is to contact the local system administrator and request that the file be restored from a backup copy. If the damage is minor, repair through editing may be possible. After such a repair, the **admin** command must be executed:

admin –z s.*file*

The purpose of this is to recompute the checksum and bring it into agreement with the contents of the file. After this command is executed, any corruption that existed in the file will no longer be detectable.