Source Library Manager 1.5
User's Guide

Jan Gray

Microsoft Applications Division
July 11, 1988

1. Introduction

SLM is a source code control system for DOS, OS/2, and UNIX environments. SLM manages a collection of files called a project; the project may be spread over a tree of directories with a single root. SLM also manages a history of changes to the project. It can retrieve old versions of project files and can display a log of changes to the project.

Each user gets his own private copy of the project. Behind the scenes there is also a "master" copy which holds the current global state of the project. Each user modifies files on his machine, and when satisfied with his changes, installs the new versions of files into the master sources. Other users may then "synchronize" to the new versions, copying them to their local copy of the project. Using this organization, it is even possible for many users to modify different areas of the same file at the same time, and later painlessly merge their changes together.

2. Sample session

To demonstrate the SLM tools, here is a sample DOS development session (output may be slightly different for UNIX users). First you must join an existing SLM administered project. To do this, you create a directory to hold your local copy of the project, and **enlist** that directory into the project. (On DOS you must have a unique volume label on your disk.)

```
C:> vol
Volume in drive C is you-C
C:> mkdir 747
C:> cd 747
C:\747> enlist -s \\boeing\SLM 747
C:\747> ls -l
d-----      0 Thu Jul 14 18:08:08 1988 [parts]
--ra--     44 Thu Jul 14 18:07:28 1988 airlines*
--ra--     71 Thu Jul 14 18:07:54 1988 features*
--rah-     87 Thu Jul 14 18:07:01 1988 slm.ini*
```

Now you are enlisted in the project. SLM created a hidden file named **slm.ini**, other files named **airlines** and **features**, and a directory named **parts**. SLM stores command initialization information in the **slm.ini** file; you can ignore it. The other files and directory are local copies of the files in the project; **parts** is a subdirectory with other SLM managed files in it. The other enlisted users also have their own local copies of these files.

At this point, the files in the project are *checked in*. They are under the control of SLM, and you may examine but not modify them. To change a file, it must be checked out first:

```
C:\747> out airlines
out: warning: modifying a released project; local version 1.50.05
C:\747> ls -l airlines
---a--      27 Thu Jul 14 18:07:28 1988 airlines*
```

SLM has made the file writable, so now you can modify it. (Ignore the warning for now; it is explained in **Versions**, below.)

```
C:\747> type airlines
United
American
Eastern
C:\747> echo Continental >>airlines
C:\747> type airlines
United
American
Eastern
Continental
```

After you complete your changes, the file is checked back into SLM.

```
C:\747> in airlines
Comment for airlines: Added Continental Airlines.
in: warning: modified a released project; now version 1.50.05
C:\747> ls -l airlines
--ra--      40 Fri Jul 15 9:01:00 1988 airlines*
```

Behind the scenes, SLM records the differences between the current master version of **airlines** and your new version ("making a diff"), copies your new version to the master sources, and makes your local copy read-only again. The comment you supply can be used months from now to help you or the other users remember why you changed the file.

Now, consider what happens if another user checks out, modifies, and checks a file back into the project (say **features**). He has the new version, and the master project sources has the new version, but you still have the old version. Your file is *out of synchronization* with the current state of the project. To resynchronize, you run the **ssync** command:

```
C:\747> ssync
ssync: warning: updating features
```

Similarly, the other users won't get a copy of your changes to **airlines** until they each run the **ssync** command.

In practice, you will find that you rarely use any SLM commands other than **ssync**, **out**, and **in**. The remainder of the SLM commands demonstrated here are used much less frequently.

To add or remove files from the project, you use the **addfile** and **delfile** commands.

```
C:\747> cat >crew
pilot
co-pilot
navigator
C:\747> addfile crew
Comment for crew: File describes standard 747 crew.
```

```
C:\747> delfile crew
Comment for crew: Didn't need to describe crew after all.
```

The **log** command lists the project history; by default it prints the last few events.

```
C:\747> log
Log for 747:
time              user    op        file          comment
...
7-12-88@12:30     admin   release                 1.50.04 beta;
7-14-88@18:07     you     enlist
7-15-88@09:01     you     in        airlines v3   Added Continental Airlines.
7-15-88@10:10     joe     in        features v2   Added new features.
7-15-88@11:10     you     addfile   crew v1       File describes standard 747 crew.
7-15-88@11:30     you     delfile   crew v2       Didn't need to describe crew after all.
```

The **status** command lists the state of each file that is checked out; the **-x** flag lists the status of all files:

```
C:\747> status -x
Status for \\C:you-c\747, owner = you
Subdirectory \, version 1.50.05:
file        local-ver    ver      status   base
airlines        3         3        in
features        2         2        in
parts           1         1        (dir)
```

The current project version number is **1.50.05**. For each file, you can see its version (**ver**) and your local version (**local-ver**); since you are in sync they are identical. See the section on **Versions** for more information.

Catsrc is used to retrieve old versions of files:

```
C:\747> catsrc airlines@v2
United
American
Eastern
```

Use **scomp** to print the differences between versions of a file. By default, it prints the changes you made since you checked the files out.

```
C:\747> out airlines
C:\747> echo Air Canada >>airlines
C:\747> scomp
---------- airlines next ----------
4a1
>Air Canada
```

Sometimes SLM will prompt you to confirm an operation, e.g. "*File* **is checked out and should be deleted; delete now?**". It is always safe to answer **n** (no). If you are writing an overnight batch script using SLM commands, and you know you won't be there to answer **no**, you can achieve the same effect by running the command with a null standard input:

```
C:\747> ssync <nul        DOS
% ssync </dev/null        UNIX
```

The **-f** (force) flag causes SLM commands to always proceed without confirmation; it is the same as always answering **yes**, and so is a little bit dangerous! Even if you forget to specify **-f** you can achieve the same effect by answering **f** (force) the next time SLM prompts for a yes/no answer.

Every SLM command responds to the **-h** (help) flag with a brief description and command line usage information.

3. File types

SLM manages a variety of types of files:

text	An ordinary ASCII text file, with lines terminated by carriage return/line feed characters (DOS or OS/2) or newlines (UNIX). Can be merged (see **Merging changes**, below). Change history maintained by SLM.
binary	A file containing non-printable characters, for instance a **.exe**. Can't be merged. Change history maintained by SLM.
unrecoverable	A file with any kind of contents; its change history is **not** maintained by SLM. When a new version of an unrecoverable file is checked in, the old one is lost forever. (But see **in -k**).
directory	Stores a subdirectory of the project.
version	A file which contains, in some format, the current project version number. Automatically updated by SLM. Can't be checked out. See **Versions**, below.

The file type is set using the **-t** flag of **addfile**.

4. Merging changes

Although it didn't happen in our sample session, it is possible for two (or even *many*) users to edit the same file at the same time. Each checks the file out; the second user will be warned:

```
C:\747> out foo
out: foo is already checked out to joe; check out anyway? y
```

When one of the users checks in his new version of the file, SLM first saves the current master source file away as a *base* file, then makes a diff and stores the new version in the project sources directory as usual.

The base file represents the starting point of the other user. Later, when he checks in his version, SLM will compute the difference between his base file and his version, and will apply those changes to the current master version, storing the result in the local directory. Thus both user's changes will be merged together in the second user's copy of the file.

If the changes conflict, (e.g. both modified the same line of the file), SLM indicates the conflict with a construction like this:

```
/-----------------------------------------------\
+++++++++++++++: Original source
original lines
+++++++++++++++: Current source
current source (checked in by first user)
```

```
++++++++++++++: User's version
second user's conflicting changes
\----------------------------------------------/
```

The second user will have to fix the conflicts by hand (using a text editor). For each conflicting change, he will keep some combination of the original lines and the changes.

Whether or not the merge was successful, SLM asks the second user to examine the merged file, because SLM can't detect changes which conflict semantically. When he has corrected or verified the change, he runs in a second time, which stores the merged file in the master sources.

5. Versions

In the sample session, **catsrc airlines@v2** was used to retrieve an old version of a file. We used **v2**, the *file version number*, to identify the version we wanted. There are many ways to refer to particular times in the project history.

5.1 Date and time

SLM logs all events which modify the project's state, such as checking in a new version of a file. Each log entry has a time stamp, and so it is possible to refer to events by date and time. These commands could retrieve the second version of **airlines** (assume that *now* is the next day, 7-16-88, at 12:00 noon):

catsrc airlines@7-15-88@8:00	(before we changed it yesterday at 9:01.)
catsrc airlines@7-15-88	(as of the first second of 7-15-88).
catsrc airlines@7-15	(defaults to current year).
catsrc airlines@.-1@8:00	(. means today, so **.-1** is yesterday).

Try to keep the clock on your machine accurate, and be wary of using SLM during the hour that the clock is changed for Daylight Savings Time!

5.2 Project version number and name

SLM automatically maintains a *project version number* of the form *major.minor*[*.revision*], such as '**1.50**'. The *revision* component is optional, and will often be omitted to identify an external release of a product. When a project has reached a milestone, the project administrator runs **sadmin release**, closing development of that version. Later on, references to that project version will refer to the project and its files as they were at the instant the project was released.

The project administrator can optionally assign a *project version name* to the project, which is a "synonym" for the project version number.

If you examine the **log** output in our sample session, you will see a line which reads

7-12-88@12:30 admin release 1.50.04 beta;

This means that the project administrator released version **1.50.04** of the project, and he named it release **beta**. Therefore either of

```
catsrc airlines@1.50.04
catsrc airlines@beta
```

will retrieve **airlines** as it was at the release of the project (before we made our changes to airlines).

When we checked in the new version of **airlines** in the sample session, we changed the state of the project. It is no longer identical to version **1.50.04**, and so SLM automatically increments the project version to **1.50.05**, and clears the project version name. From that point on, all changes made to the project will be contributing to the **1.50.05** release, until the next milestone is reached, and once more the project administrator runs **sadmin release**. This models the scheme used in several Applications Division projects.

5.2.1 Local project version number

As we can tell from the log in the sample session, the project had just been released by **admin** when we checked out **airlines**, changed it, and checked it back in. When the file was checked out, SLM warned

out: warning: modifying a released project; local version 1.50.05

SLM interpreted the check out of the file as indication that we were going to modify the project. Therefore it incremented the *local* project version number. If we had any *version* type files (see below) it would have updated them to the newer version number. Other users who ssync will still have the released project version number (**1.50.04**). The purpose of this feature is to give enlisted users who are working on a new version the newer version number as early as possible.

When we checked **airlines** back in, SLM issued another warning:

out: warning: modified a released project; new version 1.50.05

indicating that the project had changed from the released version, and the project version number was automatically incremented. The local version (**1.50.05**) is once again in agreement with the other users' version (**1.50.05**).

5.3 File version number

We have already seen that SLM associates a *file version number* with each file, This number is initialized to 1 when you add a file to the system, and is automatically incremented whenever the master file is changed, through **addfile, delfile, in**, or **sadmin rename**. The file version number syntax is *vnumber*:

```
catsrc airlines@v2
```

SLM keeps track of which version of each file was last sync'd to each enlisted directory. If a file is in sync with the project, its local file version will equal the master file version; if a file is *not* in sync, it will have a lower local file version than master version. You can use **status -x** to find out the version of each file in your enlistment.

5.4 Version.h

Developers often want to include the current version information in their programs. If you install a *version* type file in your system, using

```
addfile -tv version.h
```

then SLM will automatically maintain up to date project version information in this **.c** or **.h** file:

```
#define rmj     major
#define rmm     minor
```

```
#define rup             revision (or 0)
#define szVerName       "name"
#define szVerUser       "you"
```

If the filename ends in a **.asm** or **.inc** extension, it will get an assembler version:

```
rmj             equ     major
rmm             equ     minor
rup             equ     revision (or 0)
szVerName       equ     "name"
szVerUser       equ     "you"
```

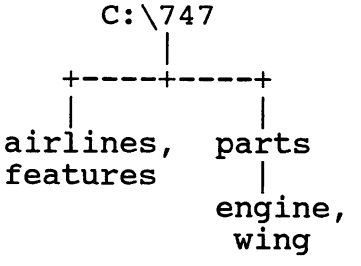
Otherwise,

major.minor.revision name you

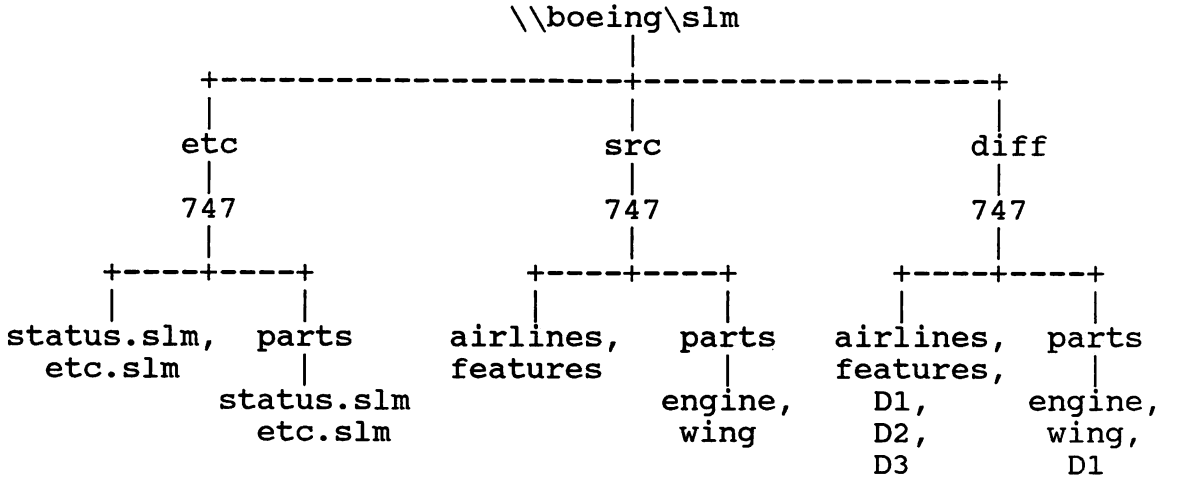
If the project version number changes, **ssync** (as well as **addfile, delfile, out, in**, and **sadmin rename**) will automatically update your *version* type files for you.

6. Directory structure

As mentioned earlier, each user gets his own copy of the project, and behind the scenes SLM keeps its own master directory structure. Here is the local structure of the project presented in section 2:



Here is the global structure:



Each of the three directories contains the same directory "skeleton". Each **etc** subdirectory contains a **status.slm** (current state of the project), a **log.slm** (history of the project), and zero or more base files (**b***number*). The **src** subdirectories contain the official, master version of each file in the project.

Diff subdirectories may contain old-style individual diff files (**D***n*), as well as the new style diff archives. These archives, one per source file in the project, contain the annotated history of each change made to the file. Here is a typical diff archive entry:

#F airlines v3	*file name and version*
#K text	*file "kind" (type)*
#O in	*SLM operation responsible for this entry*
#P 1.50.05	*project version*
#T Fri Jul 15 9:01:00 1988	*time of operation*
#A you	*author of change*
#C Added Continental Airlines	*change comment*
#I 3	*diff archive entry index*
#D 19	*beginning of diff; diff is 19 bytes long*
3a1	*body of diff*
>Continental	
newline	*extra newline*
#D 19	*end of diff*
newline	
newline	

(The extra newline ensures that even binary information (which may not end in a newline) can be stored in a diff archive entry.)

On UNIX on a common filesystem device, user's checked in files are linked to readonly copies in the master **src** directory.

7. Common problems

7.1 Changing drive label / changing network name

Sometimes you have to change a disk drive, drive label, or network name. It is possible to update your project state to the new name without defecting and enlisting. Change to the root of your local project tree, remove your **slm.ini**, and run **slmck -irs** *//server/shortname project*. This will visit each subdirectory of your copy of the project and correct the **slm.ini** file.

Note that if you plan to change *both* your drive label and your network name, you will have to repeat this operation twice (SLM needs one or the other to determine which enlisted directory you represent.) Change one of them, run **slmck** as shown above, change the other, and again run **slmck**.

7.2 Overlaid directories

It is possible for a single local directory to be simultaneously enlisted in more than one project (just run **enlist** to two separate projects). In Applications, we use this feature to overlay projects with similar directory structures; the **dosenv**, **winenv**, and **macenv** projects of **//pstools/env** are organized to allow overlaid enlisting.

The first **enlist** will create an **slm.ini** in each subdirectory. Subsequent **enlists** will *not*. Any SLM commands you run will refer to the project identified in the **slm.ini** in the current directory. If you wish to refer to a different project, you can specify the **-p** *project* on the command line, or you can use one of the **dosenv**, **winenv**, or **macenv** commands to automatically modify your **slm.ini** to refer to a different project.

Only the **slm.ini** in the current directory is read. Thus if you run a recursive command, (e.g. **ssync -r**), it will operate on the project identified in the **slm.ini** in the current directory, even if the **slm.ini**'s in subdirectories identify a different project.

If you **defect** from a project, SLM will walk your local directory tree and remove each **slm.ini** which refers to that project, *even if you are still enlisted in other projects from that directory*. To avoid problems, you must ensure that each subdirectory refers to a different project (i.e. **run macenv** in each subdirectory), or you can fix the problem after the fact by running **slmck -irs** *//server/shortname project*.

8. Commands

In this section, [] encloses optional text and | separates alternatives. *Italicized* words are variables.

8.1 Variables

project-version-name
: An alphanumeric text string identifying a particular project version.
Examples: **beta, 88000.**

project-version-number
: *major.minor*[*.revision*].
Examples: **1.50, 1.50.01.**

file-version
: **v***number,* identifying a particular version of a file. If *number* is positive, it denotes an absolute file version number; if negative it denotes a displacement off the current version (i.e. if the current version is **v7, v-2** denotes **v5**).
Examples: **v77, v-3.**

date
: Either a date (*m-d*[*-y*], (defaults to current year)) or a dot expression (.[*-days*]), where '.' denotes either *today*, or (in the context of a start date) the beginning of time.
Examples: **1-1, 12-31-88, ., .-1.**

time
: Either a date with optional time (*date*[*@h:m*[*:s*]]) or a version identifier (*@project-version-name* or *project-version-number*). If it is a date without a time, it refers to the first second of the day unless it is the end time of a time range (e.g. second parameter of **log -t** or **scomp -t**) in which case it means the last second of the day.
Examples: **1-1@1:11:11, 12-31-88@12:34, .@3:30, .-1@18:00, 1.50, 2.00.01, @beta.**

pattern
: A name containing the '?' or '*' wildcards, but no '/' or '\'.
Examples: ***.c, ?a?e?iou??y.**

filename
: A pathname to a file or directory. In some contexts this pathname must be relative and must refer to files or directories in the project. The pathname may contain the special directory symbols '.' and '..', the '/' or '\', and the '?' and '*' wildcards.

  The DOS and OS/2 versions of SLM use the familiar DOS file matching rules. On UNIX, wildcards must be quoted to protect them from the shell, but are matched using the shell's matching rules. Remember that '*.*' on UNIX SLM doesn't match **foo**, and '*' on DOS SLM doesn't match **foo.c**.
  Examples: **foo.c, ../subdir/*.h, dir*/*srcs/make????.**

filetime
: A *filename* with an optional *time* specification: *filename*[@[*time*|*file-version*]]. A *filetime* which does not specify a time refers to the current version of the file.

Examples: **foo.c, foo.c@v7, foo.c@v-2, foo.c@1-1-88, foo.c@.-1@12:30, foo.c@1.50, foo.c@beta, ../subdir/*.h@1.50.03.**

project[*/subdir*] A project name optionally followed by a subdirectory of the project. If you do not specify a particular subdirectory, SLM will use the one specified by your **slm.ini** (or / if it doesn't exist).
Examples: **proj, proj/, proj/level1/level2.**

8.2 Flags

Some option flags are common to many commands:

-& redirects *stderr* to the same place as *stdout.*

-a [*pattern*] all; recursively process all files from the top directory enlisted in the project. If a *pattern* is specified, process all files matching the pattern.

-c *comment* specify a change comment. If you don't give a comment on the command line, SLM will prompt you for one for each file.

-f forces operation; all queries are automatically answered *yes.*

-h help; print a brief description and usage information.

-r [*pattern*] recursive; process all files under the directory arguments. If no directories are specified, process all files under the current directory. If a *pattern* is specified, process all files matching the pattern.

-v verbose; all major operations performed are listed on *stderr.* **-v** causes more detailed output for **log** and **status** commands.

-s *SLM-root* specifies or overrides the current SLM root directory.

-p *project*[*/subdir*]
specifies or overrides the current project name, and optionally, the project subdirectory. Note that commands which expect a project argument (e.g. **addproj** *project*) will also accept the **-p** *project* notation.

-t *time* [*time*] specify a starting time or a time range. Note that *project-version-name*s must be prefixed by @ to avoid confusion with any subsequent file names following on the command line.

Flags may be specified together or may be space separated. A flag's arguments, if any, must immediately follow the flag, but a space before the first argument is optional. Here are some examples of how a set of flags would be interpreted:

status -fv	status -f -v
status -vf	status -v -f
status -vfp proj1	status -v -f -p proj1
status -vfpproj1	status -v -f -p proj1

status -pfvproj1 status -p fvproj1

The **-a** and **-r** flags complicate matters, because they take an optional pattern argument. A pattern must contain either the * or ? wildcard (and must not begin with a -):

status -rf status -r -f

status -rf*.* status -r f*.*

8.3 Pathname interpretation

8.3.1 Relative and absolute pathnames

Commands understand relative pathnames to project names, project files, and directories. (Absolute pathnames are not currently implemented.)

8.3.2 Specifying project and project subdirectory

You can also directly specify a particular SLM installation (**-s** *SLM-root*) and project and optional project subdirectory (**-p** *project*[*/subdir*]). Use these flags when you don't have an **slm.ini** or you wish to override it (e.g. to specify a different project in the overlaid **dosenv/winenv/macenv** installation). For commands such as **ssync** which require that you are enlisted, you must specify a project subdirectory that exactly corresponds to your current directory. For example, if you were enlisted in the overlaid **dosenv/winenv/macenv** projects, and were in subdirectory **/bin**, you could ssync to **winenv** using **ssync -p winenv** or **ssync -p winenv/bin**, but not **ssync -p winenv/**, because you aren't currently in winenv's root directory.

8.3.3 Matching wildcards against local or master directories

Local directories sometimes contain different files than the master project directories. In the context of wildcard filename expansion it is necessary to differentiate between the two locations. All wildcard pathnames are matched against files in the master directory, unless the pathname is prefixed by **./** or the command is **addfile**, in which case it is matched locally. Under UNIX, it may be necessary to quote wildcard pathnames, lest they be expanded by the shell.

8.3.4 Directory arguments and recursive flags

The SLM commands which accept *filename* arguments generally ignore directory names, unless you specify **-r**, in which case the command will recursively process directory arguments, their files and directories, etc. If you omit all pathname arguments, **-r** will affect all files in all subdirectories. **-r** also accepts a wildcard pattern argument to match against files, so that **-r *.c** matches all **.c** files in the current directory and below it. For example:

-r	All files in and below this directory.
-r dir	All files in and below **dir**.
-r *.c	All **.c** files in and below this directory.
-r *.c dir	All **.c** files in and below **dir**.
-r dir*	All files beginning with **dir** in and below this directory (!).

The **-a** (all) argument ignores pathname arguments and selects all files (or all files matching the given pattern) in the project. Like **-r**, it also accepts a *pattern* argument, which selects all files matching the pattern.

8.4 Command summary

addfile	add new files to the project.
addproj	add a new project to the SLM system.
catsrc	recreate old versions of files.
defect	disassociate your entire directory tree from a project.
delfile	delete files from the project.
delproj	remove a project from the SLM system.
enlist	join an existing SLM project.
in	check files back into the project.
log	display project history.
out	check files out for editing.
sadmin	miscellaneous SLM administration commands.
scomp	list differences between versions.
slmck	check the integrity of the SLM system.
ssync	synchronize the local files to the master project sources.
status	print current status of project files.

8.5 Command descriptions

Here follow manual pages for each SLM command.

NAME

addfile - add new files to the project

SYNOPSIS

addfile [**-&fhv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-r** [*pattern*]] [**-t** [**btuv**]] [**-c** *comment*] [*filename(s)*]

ARGUMENTS

-& redirects *stderr* onto *stdout*.

-f force operation; all queries are automatically answered *yes*.

-h help; print a brief command usage summary.

-v verbose; all major operations performed are listed on *stderr*.

-s *SLM-root* specify the current SLM root directory.

-p *project*[*/subdir*]
specify the current project and optional subdirectory.

-r [*pattern*] recursive; add all files under the directory arguments. If no directories are specified, add all files under the current directory. If a *pattern* is specified, add all files matching the pattern.

-t [**btuv**] specify the type of all files being added:

b	binary
t	text
u	unrecoverable
v	version

-c *comment* specify the reason for adding the file.

filename(s) specify the pathname(s) to the files or directories that you wish to add to the project.

DESCRIPTION

Addfile adds new files to the project. The files are copied to the SLM master source directory, and added file is recorded in the log.

Each added file is set to version 1.

If no **-t** option is given, **addfile** will guess at a file type. If a file appears to be text, it will be given the *text* type, otherwise the *unrecoverable* type. If **-tb** or **-tt** is given, SLM checks that the specified files are of the specified type.

Use **addfile -r** *directory* to add a directory and all of its contents to a project. If the **-r** flag is not specified, the directory is added to the project, but its contents are not.

EXAMPLES

addfile -tb foo.exe
add foo.exe to project as a binary file. (*Binary* files can be recovered by catsrc, *unrecoverable* files can't.)

addfile -tb *.* add all files in the current directory to the project. **Addfile** will detect text files and ask if they should be added as text files.

addfile -tv version.h
adds a *version* file to the project; this file will be created and automatically updated to

the current project version number as it changes. The specified file should not already exist.

addfile -c "my new subdirectory" -r subdir
add subdir and its contents to the project. Non-text, non-directory files will be added as *unrecoverable*.

SEE ALSO

delfile.

NAME

addproj - add a new project to the SLM system.

SYNOPSIS

addproj [-&fhv] [-s *SLM-root*] **[-p]** *project*

ARGUMENTS

-&	redirects *stderr* onto *stdout*.
-f	force operation; all queries are automatically answered *yes*.
-h	help; print a brief command usage summary.
-v	verbose; all major operations performed are listed on *stderr*.
-s *SLM-root*	specify the current SLM root directory.
-p	optional project flag (ignored).
project	the project to create.

DESCRIPTION

Addproj adds a new project to the SLM system. It creates the master project directories and the project status and log files.

EXAMPLE

addproj -s //server/shortname proj

create a project named proj in the named SLM installation at //server/shortname.

SEE ALSO

delproj.

NAME

catsrc - recreate old versions of files.

SYNOPSIS

catsrc [**-&fhv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-[a|r]** [*pattern*]] [**-x**] [**-d** *directory*] [**-t** *time*] [*filetime(s)*]

ARGUMENTS

-& redirects *stderr* onto *stdout*.

-f force operation; all queries are automatically answered *yes*.

-h help; print a brief command usage summary.

-v verbose; all major operations performed are listed on *stderr*.

-s *SLM-root* specify the current SLM root directory.

-p *project*[*/subdir*]
specify the current project name and optional subdirectory.

-a [*pattern*] all; retrieve all files from the top directory enlisted in the project. If a *pattern* is specified, retrieve all files matching the pattern.

-r [*pattern*] recursive; retrieve all files under the directory arguments. If no directories are specified, retrieve all files under the current directory. If a *pattern* is specified, retrieve all files matching the pattern.

-x retrieve the specified files or by default the entire directory as it existed at the specified time. *Time* and *pattern* arguments must not be specified with this option, and you must specify an output directory (using **-d** *directory*).

-d *directory* directory into which output files are written. The directory can be an absolute pathname, or it can be relative to the current directory. It will be created if it does not already exist.

-t *time* time from which to retrieve the files.

filetime(s) pathnames (with optional *@time* modifiers) to retrieve.

DESCRIPTION

Catsrc recreates old versions of *text* and *binary* files. Which version to recreate may be specified by the **-t** (time) flag, or by giving a filename and time using the @ modifier. For instance, foo.c (version 7, check in on 7-26-88, project version name "alpha", version number 1.3.17), could be recreated by any of these commands:

```
catsrc -t 7-26-88 foo.c
catsrc -t 1.30.17 foo.c
catsrc -t @alpha foo.c
catsrc foo.c@7-26-88
catsrc foo.c@1.30.17
catsrc foo.c@alpha
catsrc foo.c@v7
```

The **-t** option applies to *all* filename arguments that lack an @ modifier.

The **-x** flag recreates files as they existed at the specified time; this can matter if the effects of **sadmin rename** are taken into consideration. If "foo.c@v7" were renamed to be "bar.c@v8", then **catsrc**

bar.c@v3 would retrieve an earlier version, as would **catsrc -x foo.c@v3**, but **catsrc -x bar.c@v3** would *not*, because there never was a bar.c version 3. The **-x** flag is most useful for recovering an entire directory or directory tree.

By default, **catsrc** writes the old files to the standard output stream. The **-d** option specifies a directory into which the output files are written. If the **-r** (recurse) or **-a** (all) options are given, **catsrc** will recurse, creating subdirectories in the target directory as required.

Without **-r**, directory arguments are ignored.

EXAMPLES

catsrc foo.c@v-1

recreate, on *stdout*, the previous version of foo.c.

catsrc -a -x -t 1.01 -d old

recreate the entire project as it was when released as project version 1.01, in the directory old.

SEE ALSO

out -t.

NAME

defect - disassociate your entire directory tree from a project.

SYNOPSIS

defect [**-&fhkv**] [**-s** *SLM-root*] [[**-p**] *project*]

ARGUMENTS

-&	redirects *stderr* onto *stdout*.
-k	keep the local copy of the project files.
-f	force operation; all queries are automatically answered *yes*.
-h	help; print a brief command usage summary.
-v	verbose; all major operations performed are listed on *stderr*.
-s *SLM-root*	specify the SLM root directory.
[**-p**] *project*	specify the project name.

DESCRIPTION

Defect disassociates your current directory from an SLM project. Unless the **-k** (keep) option is specified, SLM will remove the local copies of project files. In any event, the directory structure is left intact.

EXAMPLE

defect -k	Remove the user from the project, retaining a copy of the local project files.

SEE ALSO

enlist.

NAME

delfile - delete files from the project

SYNOPSIS

delfile [**-&fhkv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-r** [*pattern*]] [**-c** *comment*] [*filename(s)*]

ARGUMENTS

-& redirects *stderr* onto *stdout*.

-k keep the local copy of the project files.

-f force operation; all queries are automatically answered *yes*.

-h help; print a brief command usage summary.

-v verbose; all major operations performed are listed on *stderr*.

-s *SLM-root* specify the current SLM root directory.

-p *project*[*/subdir*]
specify the current project and optional subdirectory.

-r [*pattern*] recursive; remove all files under the directory arguments. If no directories are specified, remove all files under the current directory. If a *pattern* is specified, delete all files matching the pattern.

-c *comment* specify the reason for deleting the files.

filename(s) project files to remove.

DESCRIPTION

Delfile deletes the specified files from the project's source pool. Unless the **-k** (keep) option is given, the local copies of project files will be removed.

Each file's version number is incremented as the file is deleted.

Even though the file is deleted from the current project, its history persists, and old versions of the file can still be retrieved by **catsrc** (if it is a *text* or *binary* file).

Use **delfile -r** *directory* to remove a directory and all its contents from the project. Without the **-r** flag, the directory is deleted only if it contains no project files.

EXAMPLE

delfile -c obsolete subdir/old*.c
remove from the project all files in **subdir** matching **old*.c**.

SEE ALSO

addfile, SLM Administrator's Guide (sadmin deldiff, sadmin exfile).

NAME

delproj - remove a project from the SLM system.

SYNOPSIS

delproj [-&fhv] [-s *SLM-root*] **[-p]** *project*

ARGUMENTS

-&	redirects *stderr* onto *stdout*.
-f	force operation; all queries are automatically answered *yes*.
-h	help; print a brief command usage summary.
-v	verbose; all major operations performed are listed on *stderr*.
-s *SLM-root*	specify the current SLM root directory.
-p	optional project flag (ignored).
project	specify the project to remove.

DESCRIPTION

Delproj removes a project from the SLM system. It deletes the master sources for the specified project. **Delproj** will fail unless all project members have defected from the project.

EXAMPLE

delproj -s //server/shortname oldproj
remove the oldproj project from the specified SLM installation.

SEE ALSO

addproj.

NAME

enlist - join an existing SLM project.

SYNOPSIS

enlist [-&fghv] [-s *SLM-root*] [**-p**] *project*[*/subdir*]

ARGUMENTS

-&	redirects *stderr* onto *stdout*.
-f	force operation; all queries are automatically answered *yes*.
-g	ghost; instead of making a local copy of the sources, just mark them *ghosted*.
-h	help; print a brief command usage summary.
-v	verbose; all major operations performed are listed on *stderr*.
-s *SLM-root*	specify the current SLM root directory.
-p	optional project flag (ignored).
project[*/subdir*]	specify project (and optional subdirectory tree) to enlist in.

DESCRIPTION

Enlist adds your directory to the team of directories "enlisted" in the project. **Enlist** recursively copies all project files and subdirectories to your machine, unless you specify **-g** (ghost), in which case it only copies the subdirectories.

(*Ghosted* files are treated just like ordinary checked in files, except they don't appear in your local copy of the project. You can retrieve your local copy of *ghosted* files using **ssync -u** (unghost) or by checking them out.)

Occasionally, **enlist** may fail to add a subdirectory (because another project member was using a project subdirectory at the time you ran enlist); it will print the warning **directory** *dir* **already enlisted in** *project/subdir*. If this happens, you can simply run **enlist** from your local project root as many times as necessary to add all project subdirectories.

Enlist creates a hidden file named **slm.ini** or **.slmrc** to record local project information.

EXAMPLES

enlist -s //server/shortname proj
enlist into the **proj** project in the SLM installation //server/shortname.

enlist -s //server/shortname proj/sub/subtwo
enlist only into the /sub/subtwo subdirectory tree of the project.

SEE ALSO

defect, ssync -u.

NAME

in - check files back into the project.

SYNOPSIS

in [-&bfhikouv] [-s *SLM-root*] **[-p** *project*[*/subdir*]] **[-[a|r]** [*pattern*]] **[-d**[*suffix*]=[*suffix*]] **[-c** *comment*] [*filename(s)*]

ARGUMENTS

-& redirects *stderr* onto *stdout*.

-b ignore changes in white space and tabs when generating change history. ('**-b**' comes from '**diff -b**').

-f force operation; all queries are automatically answered *yes*.

-h help; print a brief command usage summary.

-i ignore; discard any changes you made to the files since checking them out (just as if you had never checked them out).

-k checKpoint; save a copy of the current file, instead of computing a diff between the current file and the new version. (This is how *binary* files' history is maintained.) Useful for *text* and *unrecoverable* files: if you make checkpoints every few check ins, **catsrc** will be able to retrieve old versions faster since it only has to unmerge diffs back to the most recent checkpoint file in the history. It can also be used to "force" a change when checking in an otherwise unchanged text file.

-o check in all files which are checked out to this directory. This option may not be specified with *pattern* or *filename* arguments.

-u update the sources but leave the files checked out; identical to **in** *files;* **out** *files*.

-v verbose; all major operations performed are listed on *stderr*.

-s *SLM-root* specify the current SLM root directory.

-p *project*[*/subdir*]
specify the current project and optional subdirectory.

-a [*pattern*] all; check in all files below the top directory enlisted in the project. If a *pattern* is specified, check in all files matching the pattern.

-r [*pattern*] recursive; check in all files under the directory arguments. If no directories are specified, check in all files under the current directory. If a *pattern* is specified, check in all files matching the pattern.

-c *comment* specify the change comment.

-d[*suffix*]=[*suffix*]
(UNIX only). Delete files whose names are formed from the specified *filenames* by substituting the second suffix for the first. This is used to force recompilation of files dependent upon the files which are being checked in.

filename(s) the pathnames of the files or directories to check in.

DESCRIPTION

In checks the specified files back into the SLM system. It records the difference between the current source file and the file you are checking in, installs your version as the official source version, makes your local copy *read-only,* and records it as "checked in". If another user has changed the file since you checked it out, SLM attempts to merge the changes together into your local copy of the file. If changes conflict (e.g. both users modified the same line of the file), the file must be merged by hand before it can be checked in.

Checking a file in increments its file version number.

Without **-r**, directory arguments are ignored.

EXAMPLES

in -oc "bug fix 1234"
: check in all checked out files, using the comment "bug fix 1234".

in -i foo.c
: check **foo.c** back in, discarding any local changes made since it was checked out. (Use **in -i** when you wish to abandon changes and make the file checked in (*cf.* **out -c**).

SEE ALSO

out.

NAME

log - display project history.

SYNOPSIS

log [**-&afhirv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-#** [#]] [**-t** *time* [*time*]] [**-u** *user*] [*filename(s)*]

ARGUMENTS

-& redirects *stderr* onto *stdout*.

-f force operation; all queries are automatically answered *yes*.

-h help; print a brief command usage summary.

-i in only; displays only **in**, **addfile**, **delfile**, and **sadmin rename** events.

-v verbose; all major operations performed are listed on *stderr*. **-v** also selects a more detailed two line per log entry output format.

-s *SLM-root* specify the current SLM root directory.

-p *project*[*/subdir*]
specify the current project name.

-a all; print history for all subdirectories enlisted in the project.

-r recursive; print history for the directory arguments and their subdirectories. If no directories are specified, print history for the current directory and its subdirectories.

-# [#] display a range of the log entries; the numbers are displacements from the end of the log. Thus **-10 5** prints the tenth most recent to the fifth most recent log entries. If the second number is not specified, it is taken to be the last entry, so **-20** prints the last twenty entries.

-t *time* [*time*] list events which occurred between the two *time*s (the second time defaults to *now*). Use only if **-# #** is not specified.

-u *user* only list events involving *user*.

DESCRIPTION

Log displays a summary of the changes made to the project. By default, **log** displays the last ten entries.

By default, **log** prints information on all files in the project. If you specify *filename* arguments, **log** will print the history of just those files.

EXAMPLES

log -5 -i display the last five changes made to files.

log -t .-7 display changes made to the project in the past week.

log -t 1.50 1.60 display changes made between versions 1.50 and 1.60.

SEE ALSO

addfile, defect, delfile, enlist, in, sadmin.

NAME

out - check files out for editing.

SYNOPSIS

out [**-&cfhv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-**[**a**|**r**] [*pattern*]] [**-t** *time*] [*filetime(s)*]

ARGUMENTS

-& redirects *stderr* onto *stdout*.

-b broken linked; check out all "broken linked" files, that is, writable (DOS or OS/2) or unlinked (UNIX) files that you have modified without properly checking them out (e.g. due to network failure).

-c copy; discard local changes and retrieve a copy of the file as it was when first checked out.

-f force operation; all queries are automatically answered *yes*.

-h help; print a brief command usage summary.

-v verbose; all major operations performed are listed on *stderr*.

-s *SLM-root* specify the current SLM root directory.

-p *project*[*/subdir*] specify the current project and optional subdirectory.

-a [*pattern*] all; check out all files under the top directory enlisted in the project. If a *pattern* is specified, check out all files matching the pattern.

-r [*pattern*] recursive; check out all files under the directory arguments. If no directories are specified, check out all files under the current directory. If a *pattern* is specified, check out all files matching the pattern.

-t *time* check out old versions of the specified files, as they were at the specified time or project version.

filetime(s) pathnames (with optional *@time* modifiers) to check out.

DESCRIPTION

Out checks files out to your current directory. Typically SLM will fetch the latest version of the files, make them writable, and mark them as "checked out".

Sometimes you don't want to check out the latest version. If you specify any files with the *@time* modifier, or specify the **-t** *time* option, **out** will recreate old versions of the files, save them in *base* files, and check them out. Any changes made to these files will have to be *merged* into the current sources when the files are checked in.

If you must modify a file without checking it out (because the network is down), make it writable by explicitly adding write permission. (On UNIX, you must first break the link between the files and their counterparts in the source directory (lnbrk)). You may then edit as usual. Check the file out as soon as possible. **Out** detects that you have a writable checked-in file, it knows what version you had last synchronized to, and so can retroactively check out that file out for you (as if you had checked it out before you started editing).

Without **-r**, directory arguments are ignored.

EXAMPLES

out foo.c — check out **foo.c** for editing.

out -c foo.c — get a new copy of **foo.c** as it was checked out, discarding any local changes already made to **foo.c**.

out foo.c@1-1-88
check out an early version of **foo.c**; when you check it back in it will need to be merged just as if you had originally checked it out on January 1, 1988.

chattr +r foo.c; edit foo.c; *wait for server recovery* out foo.c
force write permission onto **foo.c**, edit it, and then retroactively check it out.

SEE ALSO

in.

NAME

sadmin - miscellaneous SLM administration commands

SYNOPSIS

sadmin *command* [**-&fhv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]]:

archdiff [**-ar**] [*dir(s)*]
comment [**-ar**] [**-t** *time* [*time*]] [**-#** [*#*]] [**-c** *comment*] [*file(s)*]
deldiff [**-ar**] [**-t** *time* [*time*]] [**-#** [*#*]] [*file(s)*]
deled [**-ar**] [*dir(s)*]
dump [**-ar**] [*dump-file*]
exfile [**-ar**] [*file(s)*]
install
listed [**-ar**] [*dir(s)*]
lock [**-ar**] [*dir(s)*]
lowercase [**-ar**] [*dir(s)*]
lsdiff [**-**[**a**|**r**] [*pattern*]] [**-c** *command*] [**-#** [*#*]] [**-t** *time* [*time*]] [*filename(s)*]
lssrc [**-**[**a**|**r**] [*pattern*]] [**-c** *command*] [*filename(s)*]
release [**-ar**] [**-c** *comment*] [**-n** *project-version-name*] [[*#*].[*#*][.[*#*]]] [*dir(s)*]
rename [**-c** *comment*] *file new-name*
renameproj *new-project-name*
runscript [**-ar**] [*dir(s)*]
setfv [**-**[**a**|**r**] [*pattern*]] [**-c** *comment*] *file-version-number* [*file(s)*]
setpv [**-ar**] [**-c** *comment*] [**-n** *project-version-name*] [[*#*].[*#*][.[*#*]]] [*dir(s)*]
settype [**-**[**a**|**r**] [*pattern*]] [**-c** *comment*] **-t** [**btuv**] [*filename(s)*]
tidy [**-acr**] [*dir(s)*]
trunclog [**-ar**] **-t** *time* [*time*]
unlock [**-ar**] [*dir(s)*]

ARGUMENTS

-& redirects *stderr* onto *stdout*.

-f force operation; all queries are automatically answered *yes*.

-h help; print a brief command usage summary.

-v verbose; all major operations performed are listed on *stderr*.

-s *SLM root* specify the current SLM root directory.

-p *project*[*/subdir*]
specify the project and optional subdirectory.

-a [*pattern*] all; process all files under the top directory enlisted in the project. If a *pattern* is specified, process all files matching the pattern.

-r [*pattern*] recursive; process all files under the directory arguments. If no directories are specified, process all files under the current directory. If a *pattern* is specified, process all files matching the pattern.

DESCRIPTION

Most **sadmin** commands are provided for your project administrator, but there a few that the average user should know about.

Sadmin runscript. All SLM commands have an atomicity property; they don't change the state of any files in your project until they have verified that all the new state can be copied to your disk, etc. This is implemented by copying new files to temporary files, and renaming them at the last minute. Thus it is

always safe to interrupt SLM (hit Break); if it hasn't reached the commit point, it will back out; if it has, it will soon complete renaming files and return control to you. The problem arises when a DOS user **reboots** his machine; SLM has no control over that event, and his changes are left half-installed. The next user of the project will see a message like

a previous run of SLM was terminated while processing *script files*; **run sadmin runscript for** *project*.

In that event, you should go to the appropriate subdirectory of your local project and run **sadmin runscript** as instructed.

Sadmin unlock. Sometimes you might notice that a particular user has had a project locked for a very long time. It may be that the user rebooted his machine leaving the status file locked. You should try to contact him to ensure that, indeed, he has accidently left the status file locked, and is not merely checking in many files or doing some other slow operation. When you are sure that the status file is wrongly locked, then running **sadmin unlock** will correct the problem.

Sadmin rename is used to rename a file (not directory) while still maintaining its change history. It is similar to **copy** *file newName*; **delfile** *file*; **addfile** *newName*, except that *newName* retains *file*'s change history. The current file version number of the new file is one greater than the version of the old file.

SEE ALSO

SLM Administrator's Guide, slmck.

NAME

scomp - list differences between versions

SYNOPSIS

scomp [**-&bdfhmv**] [**-s** *SLM-root*] [**-#** [*#*]] [**-p** *project*[*/subdir*]] [**-**[**a**|**r**] [*pattern*]] [**-t** *time* [*time*]] [*filename(s)*]

ARGUMENTS

-& redirects *stderr* onto *stdout*.

-b ignore changes in white space and tabs; runs **diff -b**.

-d diff between current master file and local file ("how does my version compare with the newest one?").

-f force operation; all queries are automatically answered *yes*.

-h help; print a brief command usage summary.

-m diff between base file and current master file ("what have others done to the file while it's been checked out?").

-v verbose; all major operations performed are listed on *stderr*.

-s *SLM-root* specify the current SLM root directory.

-p *project*[*/subdir*]
specify the current project and optional subdirectory.

-a [*pattern*] all; **scomp** all files under the top directory enlisted in the project. If a *pattern* is specified, **scomp** all files matching the pattern.

-r [*pattern*] recursive; **scomp** all files under the directory arguments. If no directories are specified, **scomp** all files under the current directory. If a *pattern* is specified, **scomp** all files matching the pattern.

-# [*#*] print all diff files referenced by the range of log entries; the numbers are displacements from the end of the log. Thus **-10 5** prints the tenth most recent to the fifth most recent diffs. If the second number is not specified, it is taken to be the last entry, so **-20** prints the last twenty diffs. If files are specified, the ranges apply to each file.

-t *time* [*time*] print all diff files created for changes made to the specified files between the specified times (the second time defaults to *now*).

filename(s) the filenames to print diffs for.

DESCRIPTION

Scomp lists differences between specified versions of *text* type files. It can be run whether or not the specified files are checked out, but by default, **scomp** lists the changes made to checked out files.

Differences between current local source files and current master source files are cached in a subdirectory (**slm.dif**, hidden, on DOS or OS/2, **.slmdiffs** on UNIX). **In** also checks this cache, so that the **scomp** *files*; **in** *files* idiom only needs to determine the changes made to each file once.

Without **-r**, directory arguments are ignored.

EXAMPLES

scomp display all changes made to all currently checked out files.

scomp -t 1.50.01 1.50.02 makefile
display changes made to **makefile** between the release of version 1.50.01 and version 1.50.02.

scomp -3 foo.c bar.c
print the last three diff files generated for both **foo.c** and **bar.c**.

NAME

slmck - check the integrity of the SLM system.

SYNOPSIS

slmck [-&fghilnouv] [-s *SLM-root*] [**-p** *project*[*/subdir*]] [**-[a|r]** [*pattern*]] [*filename(s)*]

ARGUMENTS

-& redirects *stderr* onto *stdout*.

-f force operation; all queries are automatically answered *yes*.

-g check the global (master) project state.

-h help; print a brief command usage summary.

-i check the SLM initialization file (**slm.ini** on DOS, **.slmrc** on UNIX).

-l check the log file (but doesn't fix it).

-n new; upgrade the project to a new SLM format.

-o override the status file lock.

-u check the user's directory (default).

-v verbose; all major operations performed are listed on *stderr*.

-s *SLM-root* specify the current SLM root directory.

-p *project*[*/subdir*]
specify the current project and optional subdirectory.

-a [*pattern*] all; check all files under the top directory enlisted in the project. If a *pattern* is specified, check all files matching the pattern.

-r [*pattern*] recursive; check all files under the directory arguments. If no directories are specified, check all files under the current directory. If a *pattern* is specified, check all files matching the pattern.

filename(s) specific files to check.

DESCRIPTION

Slmck checks either the local copy of the project (the default), or the master project directories (including the status file). If it finds any problems, it prompts for permission to correct them.

Slmck -u (fix local copy) is typically run by an SLM user to fix his project after a disaster, whereas **slmck -g** is run by the project administrator to fix the master version of the project.

Slmck is also used by the project administrator to upgrade the project to a new version of SLM.

EXAMPLES

rm slm.ini; slmck -irs *//server/shortname project*
recursively repair **slm.ini** files in this directory and its subdirectories. This is the correct thing to do if you change your network name or drive label (see previous section on **Common Problems**).

slmck -urs *//server/shortname project*
check that your local copy of the project is correct. It checks
your **slm.ini**,
that you are correctly enlisted,
that your files are in the right state (checked in or out, correctly linked, etc).

SEE ALSO
SLM Administrator's Guide, sadmin.

NAME

ssync - synchronize the local files to the master project sources.

SYNOPSIS

ssync [**-fghikuv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-**[**a**|**r**] [*pattern*]] [**-d**[*suffix*]=[*suffix*]] [*filename(s)*]

ARGUMENTS

-& redirects *stderr* onto *stdout*.

-f force operation; all queries are automatically answered *yes*.

-g "ghost" the selected files and remove your copies from your local directory.

-h help; print a brief command usage summary.

-i ignore pending merges. When these files are checked in, they will become the current versions. Other users' changes made since these files were checked out will be discarded. *Dangerous*!

-k if synchronizing to deleted files, keep the local copies.

-u "unghost" the selected files: copy them to the local directory.

-v verbose; all major operations performed are listed on *stderr*.

-s *SLM-root* specify the current SLM root directory.

-p *project*[*/subdir*]
specify the current project and optional subdirectory.

-a [*pattern*] all; synchronize all files under the top directory enlisted in the project. If a *pattern* is specified, synchronize all files matching the pattern.

-r [*pattern*] recursive; synchronize all files under the directory arguments. If no directories are specified, synchronize all files under the current directory. If a *pattern* is specified, synchronize all files matching the pattern.

-d[*suffix*]=[*suffix*]
(UNIX only). Delete files whose names are formed from the specified *filenames* by substituting the second suffix for the first. This is used to ensure recompilation of files dependent upon the newly synchronized files.

filename(s) just synchronize these files. **ssync** will report if there are other out of sync files.

DESCRIPTION

Ssync brings files in your enlisted directory up to date with corresponding files in the project's source pool. By default, **ssync** synchronizes *all* files in your enlisted directory.

Ssync -g marks the specified files ghosted and removes them from your local directory; subsequent **ssync**s will not copy them back. **Ssync -u** unghosts files ghosted by **enlist -g** or **ssync -g**.

Ssync will copy, add, delete, or merge files, or add or delete directories, as necessary, but will ask for permission to merge files.

Since **ssync** copies the out-of-synchronization master files to temporary files before installing them, it might run out of disk space unexpectedly. If this happens, **ssync** will prompt to abort the operation or continue. If you continue, **ssync** will try to copy the remaining files using a different strategy, which

won't be able to undo all of the work if aborted. Your files won't be damaged, but you might be left with a directory containing some files which are still out of synchronization. If this happens, try to free up some disk space and run **ssync** again.

Without **-r**, directory arguments are ignored.

EXAMPLES

ssync -a — Bring your copy of the entire project into synchronization.

ssync -g -r bigUselessSubdir
Ghost the contents of **bigUselessSubdir**. The directory structure will remain, but the files will be marked "ghosted" and will be removed from your local copy of the project.

ssync -d.c=.im (UNIX only). Synchronize all files; if links are made to **.c** files, remove their corresponding **.im** files.

SEE ALSO

enlist.

NAME

status - print current status of project files.

SYNOPSIS

status [**-efghlovx**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-u** *user*] [**-**[**a**|**r**] [*pattern*]] [*filename(s)*]

ARGUMENTS

-& redirects *stderr* onto *stdout*.

-e print the status of all enlisted directories, not just current directory.

-f force operation; all queries are automatically answered *yes*.

-g global status; which files are checked out, and to whom.

-h help; print a brief command usage summary.

-l list filenames (and no other information, useful for shell scripts).

-o print status of files which are checked out or out of synchronization.

-v verbose; all major operations performed are listed on *stderr*. **-v** also selects a more detailed listing of file attributes.

-x print status of all files.

-s *SLM-root* specify the current SLM root directory.

-p *project*[*/subdir*]
specify the current project and optional subdirectory.

-r [*pattern*] recursive; process all files under the directory arguments. If no directories are specified, process all files under the current directory. If a *pattern* is specified, process all files matching the pattern.

filename(s) print the status of just these filenames.

DESCRIPTION

status prints the state of the specified files. If no filenames are given, **status** prints the state of:

default checked out files

-o checked out or out of synchronization files

-x all files

in the current directory.

EXAMPLES

status *.c display status of checked out **.c** files.

status -avx display verbose local status of all files in the project.

Source Library Manager 1.5
Administrator's Guide

Jan Gray

Microsoft Applications Division
July 12, 1988

1. Administering an SLM project

This document assumes you are familiar with the **SLM User's Guide.**

1.1 Getting started

A project administrator needs a project to administrate. First you will need an SLM installation into which you will add the project. If you are starting fresh with the SLM distribution kit, you should run

install

which will prompt you for the root directory of the SLM installation. It will create **etc**, **src**, and **diff** directories in the SLM root directory, ready to store new projects.

Once SLM has been installed, you add a project to it using **addproj**:

addproj -s *SLM-root project*

This adds *project* subdirectories in the **etc**, **src**, and **diff** subdirectories of the SLM installation, and creates an empty SLM status and log file for the project in the **etc**/*project* directory.

Note that you only need to perform the above steps *once*. Now you (and your users) are free to enlist in the project, add files to it, etc. Have fun!

1.2 SLM 1.5 Conversion

If you are managing an existing project, you will have to upgrade it to the new SLM 1.5 format. (The layout of the status file has changed, and base files are now stored in the **etc** directory). The new **slmck** program will automatically do the conversion if you

*make a **backup** of your project

*run **slmck -ngrc "SLM 1.5 Upgrade"** *(**-ngrc** = = "new" "global" "recursive" "comment:")*

Slmck will first check your project as an SLM 1.3x project, then upgrade to SLM 1.5 format, then recheck your project as a SLM 1.5 project. This upgrade should be straightforward and uneventful so long as your current SLM 1.3x project can **slmck** (1.3x) cleanly.

Once you have done this, your users should obtain the new SLM executables. The old SLM binaries will not recognize the new status file format and will issue warnings like

status file for *project* has been damaged; run slmck for *project*

which users should *not* do. (They should install the new binaries!)

1.2.1 Diff archival

Instead of keeping each diff in the change history in a separate diff file, SLM 1.5 keeps change history in *diff archives*. These diff archives offer the advantages of reduced file system fragmentation, reduced directory search time, and "diff coherency": all diffs for a file are located together. The annotations accompanying each diff archive entry will make possible future source control tools (e.g. "print each line of code and when it was changed, who changed it, why it was changed, etc.").

SLM 1.5 stores new change history in diff archives, but it can still access the old SLM 1.3x history in the old style separate diff files. If you want to gain the benefits of diff archives for your old SLM 1.3x change history, you can use **sadmin archdiff** to archive the existing diffs:

*make a **backup** of your project

*run **sadmin archdiff -r**

This will search through the log for references to diff files and move the diffs to diff archives. *Once archived, the old diff files will be removed.*

For most projects, this will result in diff directories with many fewer files. Some projects have diff directories which contain literally tens of thousands of files. On DOS, OS/2 and UNIX, these directories will be sparse, with many empty slots for the now deleted diffs. To compact these directories and incur the most speed benefits you are advised to tree copy the diff subdirectory to another location (making a copy with compacted directories) and rename the copy in place of the original. Make sure your tree copy operation preserves modification times and copies empty directories!

1.3 Day to day responsibilities

SLM is supposed to be fairly maintenance free, but some problems can arise. User's clocks are sometimes incorrectly set, occasionally the server runs low on filespace, file systems can "spontaneously" get bad blocks, etc. It is a good practice to run **slmck -gr** (recursively check global state) every once in a while, to check that all files are setup correctly and nothing has gone haywire.

1.4 Common problems

User's common problems are discussed in the **SLM User's Guide.**

1.4.1 Moving an SLM installation

Sometimes you have to move an SLM installation to another server. All you need to do is tree copy the entire installation to the new location, and for each project, have each enlisted user run

```
cd root of user's enlistment
rm slm.ini
slmck -irs //new-server/new-name project
```

which will adjust their **slm.ini** files to refer to the new location of the project. (Use **sadmin listed** to determine which users are enlisted, and from which machines.)

1.4.2 Status file "stuck"

Sometimes, due to flakey network hardware/firmware, a status file gets stuck, giving permanent "error locking *status file*" error messages. Someone is holding an operating system lock on the file. If you are talking to a DOS server, use the **rserver file** commmand to determine which user is locking the file. We have found that rebooting the offending user's machine and reestablishing their connection to the server clears the lock, although sometimes it is necessary to reboot the server!

2. Commands

In this section, [] encloses optional text and | separates alternatives. *Italicized* words are variables.

2.1 Variables

project-version-name
: An alphanumeric text string identifying a particular project version.
Examples: **beta, 88000.**

project-version-number
: *major.minor*[*.revision*].
Examples: **1.50, 1.50.01.**

file-version
: **v***number,* identifying a particular version of a file. If *number* is positive, it denotes an absolute file version number; if negative it denotes a displacement off the current version (i.e. if the current version is **v7**, **v-2** denotes **v5**).
Examples: **v77, v-3.**

date
: Either a date (*m-d*[*-y*], (defaults to current year)) or a dot expression (.[*-days*]), where '.' denotes either *today*, or (in the context of a start date) the beginning of time.
Examples: **1-1, 12-31-88, ., .-1.**

time
: Either a date with optional time (*date*[*@h:m*[*:s*]]) or a version identifier (*@project-version-name* or *project-version-number*). If it is a date without a time, it refers to the first second of the day unless it is the end time of a time range (e.g. second parameter of **log -t** or **scomp -t**) in which case it means the last second of the day.
Examples: **1-1@1:11:11, 12-31-88@12:34, .@3:30, .-1@18:00, 1.50, 2.00.01, @beta.**

pattern
: A name containing the '?' or '*' wildcards, but no '/' or '\'.
Examples: ***.c, ?a?e?iou??y.**

filename
: A pathname to a file or directory. In some contexts this pathname must be relative and must refer to files or directories in the project. The pathname may contain the special directory symbols '.' and '..', the '/' or '\', and the '?' and '*' wildcards.

  The DOS and OS/2 versions of SLM use the familiar DOS file matching rules. On UNIX, wildcards must be quoted to protect them from the shell, but are matched using the shell's matching rules. Remember that '*.*' on UNIX SLM doesn't match **foo**, and '*' on DOS SLM doesn't match **foo.c**.
  Examples: **foo.c, ../subdir/*.h, dir*/*srcs/make????.**

filetime
: A *filename* with an optional *time* specification: *filename*[@[*time*|*file-version*]]. A *filetime* which does not specify a time refers to the current version of the file.
Examples: **foo.c, foo.c@v7, foo.c@v-2, foo.c@1-1-88, foo.c@.-1@12:30, foo.c@1.50, foo.c@beta, ../subdir/*.h@1.50.03.**

project[*/subdir*] A project name optionally followed by a subdirectory of the project. If you do not specify a particular subdirectory, SLM will use the one specified by your **slm.ini** (or / if it doesn't exist).
Examples: **proj, proj/, proj/level1/level2.**

2.2 Flags

Some option flags are common to many commands:

-& redirects *stderr* to the same place as *stdout.*

-a [*pattern*] all; recursively process all files from the top directory enlisted in the project. If a *pattern* is specified, process all files matching the pattern.

-c *comment* specify a change comment. If you don't give a comment on the command line, SLM will prompt you for one for each file.

-f forces operation; all queries are automatically answered *yes.*

-h help; print a brief description and usage information.

-r [*pattern*] recursive; process all files under the directory arguments. If no directories are specified, process all files under the current directory. If a *pattern* is specified, process all files matching the pattern.

-v verbose; all major operations performed are listed on *stderr.* **-v** causes more detailed output for **log** and **status** commands.

-s *SLM-root* specifies or overrides the current SLM root directory.

-p *project*[*/subdir*]
specifies or overrides the current project name, and optionally, the project subdirectory. Note that commands which expect a project argument (e.g. **addproj** *project*) will also accept the **-p** *project* notation.

-t *time* [*time*] specify a starting time or a time range. Note that *project-version-name*s must be prefixed by @ to avoid confusion with any subsequent file names following on the command line.

Flags may be specified together or may be space separated. A flag's arguments, if any, must immediately follow the flag, but a space before the first argument is optional. Here are some examples of how a set of flags would be interpreted:

status -fv	status -f -v
status -vf	status -v -f
status -vfp proj1	status -v -f -p proj1
status -vfpproj1	status -v -f -p proj1
status -pfvproj1	status -p fvproj1

The **-a** and **-r** flags complicate matters, because they take an optional pattern argument. A pattern must contain either the * or ? wildcard (and must not begin with a -):

status -rf	status -r -f
status -rf*.*	status -r f*.*

2.3 Command descriptions

Here follow administrator's manual pages for **sadmin** and **slmck.**

NAME

sadmin - miscellaneous SLM administration commands

SYNOPSIS

sadmin *command* [**-&fhv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]]:

archdiff [**-ar**] [*dir(s)*]
comment [**-ar**] [**-t** *time* [*time*]] [**-#** [*#*]] [**-c** *comment*] [*file(s)*]
deldiff [**-ar**] [**-t** *time* [*time*]] [**-#** [*#*]] [*file(s)*]
deled [**-ar**] *enlisted-directory* [*dir(s)*]
dump [**-ar**] [*dump-file*]
exfile [**-ar**] [*file(s)*]
listed [**-ar**] [*dir(s)*]
lock [**-ar**] [*dir(s)*]
lowercase [**-ar**] [*dir(s)*]
lsdiff [**-**[**a**|**r**] [*pattern*]] [**-c** *command*] [**-#** [*#*]] [**-t** *time* [*time*]] [*filename(s)*]
lssrc [**-**[**a**|**r**] [*pattern*]] [**-c** *command*] [*filename(s)*]
release [**-ar**] [**-c** *comment*] [**-n** *project-version-name*] [[*#*].[*#*][.[*#*]]] [*dir(s)*]
rename [**-c** *comment*] *file new-name*
renameproj *new-project-name*
runscript [**-ar**] [*dir(s)*]
setfv [**-**[**a**|**r**] [*pattern*]] [**-c** *comment*] *file-version-number* [*file(s)*]
setpv [**-ar**] [**-c** *comment*] [**-n** *project-version-name*] [[*#*].[*#*][.[*#*]]] [*dir(s)*]
settype [**-**[**a**|**r**] [*pattern*]] [**-c** *comment*] **-t** [**btu**] [*filename(s)*]
tidy [**-acr**] [*dir(s)*]
trunclog [**-ar**] **-t** *time* [*time*]
unlock [**-ar**] [*dir(s)*]
undump [**-ar**] [*dump-file*]

ARGUMENTS

-& redirects *stderr* onto *stdout*.

-f force operation; all queries are automatically answered *yes*.

-v verbose; all major operations performed are listed on the screen.

-s *SLM root* specify the current SLM root directory.

-p *project*[*/subdir*]
specify the project and optional subdirectory.

-a [*pattern*] all; process all files in all directories of the project. If a *pattern* is specified, process all files matching the pattern.

-r [*pattern*] recursive; process all files under the directory arguments. If no directories are specified, process all files under the current directory. If a *pattern* is specified, process all files matching the pattern.

DESCRIPTION

Sadmin commands fall into a number of different categories.

File operations **rename settype**

Status file operations
deled dump listed lock lowercase tidy undump unlock

Project history operations
archdiff comment deldiff exfile lsdiff trunclog

Version control
release setfv setpv

Miscellanous project maintenance
lssrc renameproj runscript

FILE OPERATIONS

sadmin rename [**-&fv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-c** *comment*] *file new-name*

Rename is used to rename a file (not directory) while still maintaining its change history. It is similar to **copy** *file newName*; **delfile** *file*; **addfile** *newName*, except that *newName* retains *file*'s change history. The current file version number of the new file is one greater than the version of the old file.

sadmin settype [**-&fv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-**[**a**|**r**] [*pattern*]] [**-c** *comment*] **-t** [**btuv**] [*filename(s)*]

Settype changes the current file type of the specified files. (Recall that the initial file type is set when the file is added to the system using **addfile**). If you change a *text* or *binary* file to another type, you will be unable to access its old change history, unless you change it back. Of course, if you change a file from *unrecoverable* to *text* or *binary*, it doesn't magically acquire a history of its past changes.

If you occasionally want to save the history of an unrecoverable file, you should use **in -k**, not **settype -tb** *file;* **in** *file;* **settype -tu** *file*.

STATUS FILE OPERATIONS

sadmin deled [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] *ed*

Deled removes an enlisted directory from the status file. It is useful for those circumstances when a user is unable to defect because he has reformatted his disk or some other tragedy. *Ed* may be the the owner of the directory, the *exact* directory path printed by **sadmin listed**, or the index of the enlisted directory (ied) printed by **sadmin listed**. **Deled** will prompt if a user is specified and he owns more than one directory. For recursive **deled**, specifying the owner or the directory path is preferable, because subdirectories may not conform to their parent directories. (In a subdirectory, a particular ied might refer to a different enlisted directory.)

sadmin dump [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [*dump-file*]

Dump is used to print a human (*and* **sadmin undump**) readable representation of the status file. It prints the status file header, per file information, per enlisted directory information, and the state of each file in each enlisted directory. If *file* is not specified, **dump** writes to *stdout*. Note that **dump** output is extremely version dependent.

sadmin listed [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [*dir(s)*]

Listed lists the directories enlisted in this subdirectory of the project.

sadmin lock [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [*dir(s)*]

Lock locks the current project directory against modification by other users. This is useful if you are going to do extensive administrative work and don't wish users to access the project half-way through your modifications. If a user tries to run an SLM command which might change the status file, the command will warn

status file locked by administrator *you*

The lock must be removed by **sadmin unlock**.

sadmin lowercase [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [*dir(s)*]

Lowercase will change all user, path, and file names in the directory to lowercase. It is occasionally useful for canonicalizing a set of names, particularly when accessing a networked project from both DOS and XENIX machines.

sadmin tidy [**-&acfrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [*dir(s)*]

Tidy tidies up the project. It removes unreferenced base files, removes unused file slots, and sorts the enlisted directories in the status file.

The **-c** (check) flag, which is only useful with the **-a** or **-r** flags, checks that the users enlisted in subdirectories are the same as the users enlisted in the parent directory, and prints any that are missing or extra. (Since users can enlist in a particular subtree of the project, it is not necessarily wrong for a subdirectory to have more users enlisted than its parent directory.)

sadmin undump [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] *dump-file*

Undump is used to load a previously **sadmin dump**'d file. This is used by "SLM wizards" to make otherwise difficult modifications to status files broken beyond **slmck**'s ability to repair. Use at your own risk!

sadmin unlock [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [*dir(s)*]

Unlock is used for two different purposes. One is to unlock a project which a user has accidently left locked by rebooting his machine at an inopportune time. You should try to contact him to ensure that, indeed, he has accidently left the status file locked, and is not merely checking in many files or doing some other slow operation. Only then should you run **unlock**.

The other use of **unlock** is to remove the administration lock that you placed on the project directory when you ran **sadmin lock**.

PROJECT HISTORY OPERATIONS

sadmin archdiff [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [*dir(s)*]

Archive separate old style (**D***n*) diff files into diff archives. Once archived, the old diff files will be removed. This command can even be used to *merge* old style diff files into existing diff archives (e.g. if later you restore older diff files from backups and want them to be integrated into the existing diff archives).

sadmin comment [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-t** *time* [*time*]] [**-#** [*#*]] [**-c** *comment*] [*file(s)*]

Comment changes a range of check-in comments in the log. You might use it to add or correct comments as your project matures. The range can be specified using times (**-t**) or log entry counts (**-#**). By default, all check-in comments in the range are changed, but you can limit the effects to particular files' comments by specifying *file* arguments.

sadmin deldiff [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-t** *time* [*time*]] [*file(s)*]

Deldiff removes (from the **diff** directory) old style diff files referenced by the log in the specified time range. Specify *file(s)* to delete particular files' diffs. The log entries themselves remain. This is useful for retaining change history but reclaiming space: permanently back up files in the **diff** directory then **deldiff** the files older than (say) six months. You can always recover an old version of a file by determining which diffs are referenced in the log (**sadmin lsdiff**), restoring those from the backup, then running **catsrc**.

Deletion of diff archive entries is not yet implemented. (Of course this should not be a problem for some time because archived diffs are much more space efficient than separate diff files.)

sadmin exfile [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] *file(s)*

Exfile is used to expunge a file, any associated old style diff files, and its log entries. Once a file is expunged, it will be as if the file *never* existed. Be careful.

Deletion of diff archive entries is not yet implemented.

sadmin lsdiff [**-&fv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [-[**a**|**r**] [*pattern*]] [**-c** *command*] [**-#** [#]] [**-t** *time* [*time*]] [*filename(s)*]

Lsdiff prints the full path to each diff file in the range specified by **-#** or **-t** (and optionally limited to the specified source *filename(s)*). If **-c** *command* is given, the output will be prefixed by *command*; this is suitable for generating a batch script to operate on the specified diff files.

sadmin trunclog [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] **-t** *time* [*time*]

Trunclog removes log entries in the specified range, but does not affect any referenced diff and copy files.

VERSION CONTROL

sadmin release [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-c** *comment*] [**-n** *project-version-name*] [[#].[#][.[#]]] [*dir(s)*]

Release marks a milestone in the development process. The directory is marked *released* and a release entry (including the current project version name and number) is written to the log. Later, **catsrc** will be able to recover project files (in the state that they were released) by project version name or number.

The *#.#.#* and **-n** *name* flags can be used to set the released project version number or name; otherwise the current values will be used. Each # component of '[[#].[#][.[#]]]' can be:

omitted	no change to that component.
number	set component to *number.*
+number	add *number* to component.

Here are some examples of the *#.#.#*. (Assume the current project number is **1.02.03**)

...	1.02.03	
3.2.1	3.02.01	
3.2	3.02	(optional third component defaults to **.0**)
3.2.	3.02.03	(third component is *omitted*)
+1.+2.+3	2.04.06	
.+1.0	1.03	(recall that a *revision* of 0 is not printed)

See also: **sadmin setpv**, **Versions** section of **SLM User's Guide**.

sadmin setfv [**-&fv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [-[**a**|**r**] [*pattern*]] [**-c** *comment*] *file-version-number* [*file(s)*]

Setfv changes a file's version number. Remember that a file version number is automatically incremented when the master file is changed, such as by **in**. **Setfv** can be used to advance or retard this number for whatever reason.

If you lower a file's version number, you will lose the ability to retrieve certain versions of the file by file version number. For instance, if **foo.c** is currently version 10, and you run **sadmin setfv 5 foo.c**, you will be unable to retrive **foo.c@v7**, although you can still retrieve it by date or perhaps by project version number or name. **Setfv** warns you if this occurs.

sadmin setpv [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-c** *comment*] [**-n** *project-version-name*] [[#].[#][.[#]]] [*dir(s)*]

Setpv sets the current project version number; it has no immediate effect on the log, but is used to change the current version number in anticipation of a subsequent **release** command. For instance, if

you know that the project was to be released as version **1.50**, you might use **setpv** to set the current project version number to **1.50** before **release**. This causes user's *version* type files to be updated the next time they run **ssync**.

See the description of **sadmin release** for the syntax of the *#.#.#* option.

MISCELLANEOUS PROJECT MAINTENANCE

sadmin lssrc [**-&fv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-**[**a**|**r**] [*pattern*]] [**-c** *command*] [*filename(s)*]

Lssrc prints the path to the master version of the specified source files. If **-c** *command* is given, the output will be prefixed by *command*; this is suitable for generating a batch script to operate on the specified source files.

sadmin renameproj [**-&fv**] [**-s** *SLM-root*] [**-p** *project*] *new-project-name*

Renameproj (you guessed it) changes the name of the project. Recall that the SLM installation has **etc**, **src**, and **diff** directories, each of which contains a subdirectory with the name of your project. The effect of **renameproj** is to change the names of these subdirectories to the new name. Each user will then have to run

slmck -irs *SLM-root new-project-name*

from their root enlisted directory to update the project field of their **slm.ini** files (**.slmrc** on UNIX).

sadmin runscript [**-&afrv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [*dir(s)*]

SLM keeps script files to do "atomic" changes to the project (two-phase commit). First, operations are done on temp files and logged in a script file; permanent changes are not made until the commit point, when the script is run with user interrupts deferred. Sometimes when this is happening, impatient users reboot their machine. The next time an SLM command is run in that directory, the user is issued the fatal error

a previous run of SLM was terminated while processing *script* **files; run sadmin runscript for** *project*.

Running **sadmin runscript** will fix the problem by either aborting the commands in the script (if the user hadn't reach the commit point) or rerunning the script to completion, one way or the other ensuring the project is in a consistent state.

(This is background information which you might like to know, but is not necessary to successfully run **sadmin runscript**). SLM keeps three kinds of script files. Operations which affect the user's local files are kept in a **local.scr** file in their directory. (The presence of this file will affect only that user's work; other users can still do work using their own **local.scr** files). The **ssync** command has special script files named *number*.**scr** in the **etc**/*project*/*subdir* directory. The *number* is the index of a user's enlisted directory, as printed by **sadmin listed**. **Ssync** can be run by many users simultaneously because each gets his own unique script file . The third and most common type of script file is the master script file, named **all.scr** in the **etc**/*project*/*subdir* directory. This is used by commands such as **in** for operations that affect the master project files.)

SEE ALSO

slmck.

NAME

slmck - check the integrity of the SLM system.

SYNOPSIS

slmck [**-&fghilnouv**] [**-s** *SLM-root*] [**-p** *project*[*/subdir*]] [**-**[**a**|**r**] [*pattern*]] [*filename(s)*]

ARGUMENTS

-& redirects *stderr* onto *stdout*.

-f force operation; all queries are automatically answered *yes*.

-g check the global (master) project state.

-h help; print a brief command usage summary.

-i check the SLM initialization file (**slm.ini** on DOS, **.slmrc** on UNIX).

-l check the log file (but doesn't fix it).

-n new; upgrade the project to a new SLM format.

-o override the status file lock.

-u check the user's directory.

-v verbose; all major operations performed are listed on the screen.

-s *SLM-root* specify the current SLM root directory.

-p *project*[*/subdir*]
specify the current project and optional subdirectory.

-a [*pattern*] all; check all files in all directories of the project. If a *pattern* is specified, check all files matching the pattern.

-r [*pattern*] recursive; check all files under the directory arguments. If no directories are specified, check all files under the current directory. If a *pattern* is specified, check all files matching the pattern.

filename(s) specific files to check.

DESCRIPTION

Slmck checks either the local copy of the project (the default), or the master project directories (including the status file). If it finds any problems, it prompts for permission to correct them.

Slmck -u (fix local copy) is typically run by an SLM user to fix his project after a disaster, whereas **slmck -g** is run by the project administrator to fix the master version of the project.

Slmck -gn is used by the project administrator to upgrade the project to a new version of SLM. It **slmck**s the project as version 1.3x, upgrades the status file to the SLM 1.5 format, then **slmck**s the project as version SLM 1.5.

SEE ALSO

sadmin.