E U R O P E A N   S O U T H E R N   O B S E R V A T O R Y

Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral

Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

VERY LARGE TELESCOPE

VLT Software

Configuration Management Module

User Manual

Doc.No. VLT-MAN-ESO-17200-0780

Issue 1.3

Date 23/04/01

Prepared	P. Forstmann	23/04/01	
	Name	Date	Signature
Approved	G. Filippi		
	Name	Date	Signature
Released	G. Raffi		
	Name	Date	Signature

Change Record

Issue/Rev.	Date	Section/Page affected	Reason/Initiation/Document/Remarks
1.2	10/03/96	Many	Updated for branching and client/server implementation.
1.1	21/10/96	All	Updated for new implementation (RCS only)
1.0	20/02/95		

1 INTRODUCTION

The software decribed in this manual is intented to be used in the ESO VLT Project.

1.1 PURPOSE

This document is the User Manual of the code configuration system developed for the VLT Software project.

This manual documents both end-user commands and commands reserved to the Software Configuration Control Manager (SCCM).

The manual assumes that the reader is a software developer having a basic knowledge of UNIX and familiar with software development activities applying rules described in [2]

In addition to the **Introduction** section, this manual contains two major sections:

User's Guide. Describes the features of the configuration management module and includes examples of "how to use it"

Reference. Describes all the commands available to the user.

1.2 SCOPE

This manual describes version 1.69 and higher of the Configuration Management Module.

This system is a very basic one, handling only code modules. It does not encompass all software development actvities (for example, documentation). It even does not implement the complete code configuration system as described in [1].

1.3 REFERENCE DOCUMENTS

[1] VLT-SPE-ESO-17200-0704, 1.0 / prep 1.0 02 / 09 / 94 -- Software Configuration Management System - Functional Specification

[2] VLT-PRO-ESO-10000-0228, 1.0 10 / 03 / 93 -- VLT Software, Programming Standards

[3] VLT-MAN-ESO-17200-0793 -- VLT Software, CMM Maintenance Manual

1.4 ABBREVIATIONS AND ACRONYMS

The following abbreviations and acronyms are used in this document::

CASE	Computer Aided Software Engineering
CMM	Configuration Management Module
DBMS	Database Management System
HW	Hardware
I / O	Input / Output
LCU	Local Control Unit
N / A	Not Applicable
TBC	To Be Confirmed
SW	Software

TBD	To Be Defined
VLT	Very Large Telescope
WS	Workstation
SPR	Software Problem Report
SCCM	Software Configuration Control Manager

1.5 GLOSSARY

configuration item

a collection of elements treated as a unit, for the purpose of configuration management.

module

in this document has the meaning of software module, as defined in [2].

to check out an item

to retrieve from an archive an item in order to modify it or not.

to check in an item

to archive an item after having checked it out.

1.6 STYLISTIC CONVENTIONS

The following styles are used:

bold
in the text, for commands, filenames, pre/suffixes as they have to be typed.

italic
in the text, for parts that have to be substituted with the real content before typing.

`teletype`
for examples.

`<name>`
in the examples, for parts that have to be substituted with the real content before typing.

bold and *italic* are also used to highlight words.

2 USER'S GUIDE

This chapter provides a functional description of the Configuration Management System.

2.1 OVERVIEW

The Configuration Management System gives the software developer the possibility to manage its software modules according to the following rules of configuration management:

- module items can be identified
- module is built from a consistent set of components
- module items are available and accessible
- module items never get lost (e.g. after media failure or operator error)
- changes do not get lost (e.g. through simultaneous updates)
- it is always possible to go back to a previous version
- a history of changes is kept, so that it is always possible to know who did what and when

2.2 BASIC CONCEPTS

2.2.1 Module identification

A module is identified by its name and its version number. The version number has the format <a>.<b> or <a>.<b>.<c>.<d> (<a>, <b>, <c> and <d> being positive integers). The version number is changed everytime a module is checked in: <a>.<b> becomes <a>.<b+1> and <a>.<b>.<c>.<d> becomes <a>.<b>.<c>.<d+1>.

2.2.2 Module handling

Only the SCCM can create modules using the `cmmCreate` command; the CMM first version usually starts at 1.0.

The software developer can modify the module by using the `cmmModify` command or get a read only copy of the module by using the `cmmCopy` command. To archive the module, he must use the `cmmArchive` command which will increment the version number.

It is not possible that the same module version is modified by two differents users. Once a module is under modification, only the user having issued the `cmmModify` command can modify it. However it is possible to create a branch (see section 2.5).

2.3 A DAY IN A LIFE OF A SOFTWARE DEVELOPER

The basic loop of the software developer is made of the following steps:

- checking out a module for modification
- modifying the module and testing it
- preparing the module for archiving
- archiving the module.

2.3.1 Modifying a module

To modify a module in order to correct bugs and/or to add new features, the software developer simply executes the command `cmmModify`:

```
$ cmmModify read
(... output for every directory and file retrieved ... )
Modifying read 1.3
```

`cmmModify <module>` creates in the current directory a directory `<module>` representing the module: the contents of that directory is the set of the files tagged with version `<a>.<b>+` representing the last version of the files belonging to this module (the meaning of '+' is: under modification). This tagging is done using the following string "@(#) Id" (also known as the RCS keyword) which is processed by CMM.

The user is now free to modify the whole module: files can be updated, added or deleted, directories can be added or deleted.

If the user wants to release the module because somebody else needs to modify the same version, he should use the `cmmCancel` command.

2.3.2 Retrieving the version of a module

The module version number can be retrieved using the `$Revision$` pattern. For example in C, to add a function printing the version number of the module whom it belongs to, write:

```
void xxxGetVersion()
{
          printf("xxx version: $Revision$\n",%s);
}
```

After archiving, `cmm` will substitute the right version: the function will look like this:

```
void xxxGetVersion()
{
          printf("xxx version: $Revision: 1.2 $\n",%s);
}
```

Note: if you want only the version number, you must filter the '$Revision <a>.<b> $' string.

2.3.3 Reserved patterns

The following string patterns are **always** modified in **any** file managed by CMM, whether they are lower case or upper case. They should not be used because CMM will substitute them strings linked to the status of the file.

- $author$
- $date$
- $header$

- id
- $locker$
- log
- $name$
- $RCSfile$
- $revision$
- $source$
- $state$

2.3.4 Preparing the archiving

Before archiving the module, the user should run the command `cmmCheckForArchive (in any case` this command will be automatically run by the cmmArchive):

```
$ cmmCheckForArchive read
DIRECTORIES:

src/RCS ADDED

FILES:

read/include/f~ ADDED
src/f.c ADDED
src/readonly ADDED

Add write permissions for you on:

  read/src/readonly

Delete :

  read/include/f~
Delete the links:

  read/src/lf.c

Add RCS keyword in the following files (if not binary):
read/lib/libread.a
read/object/read.da
```

This command has three steps:

1. lists modified files as well as files and directories added or deleted.
2. checks the user write access on his own files, tells to delete some backup files or some links.
3. checks whether each file has the RCS keyword.

Although CMM can handle binary files, the user should run `make clean` before archiving its module: every file which can be generated from a source file should not, in principle, be archived: this is true for man pages, object files, libraries and binaries.

CMM cannot archive links: links should be created by running `make` and deleted by running `make clean`.

`cmmCheckForArchive` fails if it detects some files in the second step : unsufficient write access or existence of backup files or existence of links.
`In general, cmmCheckForArchive` fails if the following file names types are found:
`RCS, SCCS, *~, #*, .#*, *,v*, *.old, *.bak, *.BAK, *.orig, *.a , *.o, *.tar, *.Z, *.sl, *.so, .nfs*, core, * * (file names containing blanks).`
`cmmCheckForArchive` does not fail if some files don’t have the RCS keyword because there is no easy way to classify binary files.

2.3.5 Archiving a module

To archive a module you only need to run the `cmmArchive` command giving the module name and a comment summing up the modifications:

```
$ cmmArchive read “SPR 959999 & added readInt and readFloat”
< ... cmmCheckForArchive output ... >
<  ... output for each archived file ...>
Module read archived in version 1.4
```

If `cmmCheckForArchive` fails, the module will not be archived.
Warning: try to avoid special character in the comment line; the comment line must be less than 180 characters.

CMM does not support a file based archive, all module files are archived whether they have been modified or not. This means that for a given module version each file of the module has always the same version number: i.e. the version of the module.

After having archived a module, the module directory is updated and changed into a read only one. To modify again the module, delete the module directory and run the `cmmModify` command.

At least, a module should be archived everytime it is made available to other people to be integrated or used. In addition, a module should be archived everytime the software developer wants to make a checkpoint, i.e. to be able to retrieve exactly what has been done for a given modification (SPR correction or new feature added). There are no restrictions on the number of times a module can be archived. In other words, there is no direct link between a module archiving and a module installation under INTROOT or VLTROOT.

2.4 WARNING

CMM is based on RCS which does not have any atomicity feature with respect to modification to a group of files. There are six commands which modify the archive module: `cmmModify`,`cmmCancel`, `cmmArchive`, `cmmBranch`, `cmmArchiveBranch` and `cmmCancelBranch`. In order to minimize the risk of having a inconsistent archive module, CMM does the following:

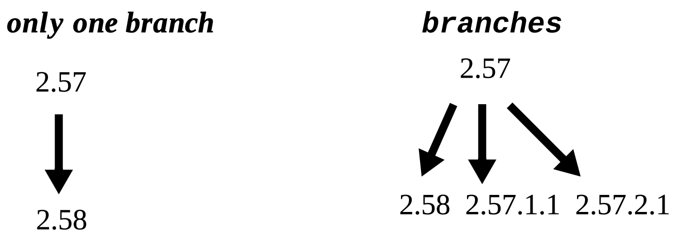
- during execution of a critical command, INTR, QUIT and SUSPEND signals are disabled.
- before doing any modification, CMM backs up the archive module.
- in case of internal error, CMM will restore automatically the archive module enabling the user to resubmit his command.

You should NEVER kill cmmModify, cmmArchive, cmmCancel, cmmBranch, cmmArchive-Branch, cmmCancelBranch, cmmCopy or cmmCompare because doing so will leave the repository in a inconsistent state. In this case, CMM does nothing and the user shall report it to the SCCM for corrective action (there will be at least a lock to remove by hand).

2.5 Branching

2.5.1 What is branching ?

It is the possibility for a file to have a tree of version instead of one single trunk. Without branching, each file version has only one descendant, the next version. With branching, each file version may have several descendants.



There is always one branch called the (main) trunk : in the above example, version 2.57 and 2.58 belong to the main trunk. But version 2.57.1.1 belong to branch 2.57.1 and version 2.57.2.1 belong to branch 2.57.2. Branches allow parallel development and especially urgent bug fixing when work on the next release has already started and is not compatible with bug fixing. When branching is used, there is no more a unique last version: there is instead a unique last for each branch.

Branching must be carefully used in order to avoid ending with many incompatible versions for the same file. Branching should only be used to fix urgent bugs and branches should always be merged in the trunk as soon as possible.

2.5.2 How to create a branch

A branch is created in 3 steps by running:

- `cmmBranch <module> <a.b>` where <module> is the module name and <a.b> is the root version for branching. This command is similar to `cmmModify` in the sense that it registers the intention to create a branch. The branch identifier is allocated by `cmmBranch` and follows the pattern <a.b>.<c> where <c> starts at 1.
- `cmmCheckForArchiveBranch <module>` is similar to cmmCheckForArchive: it displays which directories and files are added, removed or modified.
- `cmmArchiveBranch <module>` is similar to cmmArchive: it registers a new version of the module <a.b.c.1> and creates the branch <a.b.c>.

If you run `cmmCancelBranch <module>` instead of `cmmArchiveBranch`, no new version is created and no branch is created.

2.6 Working with branches

Once a branch is created, you can use `cmmModify`, `cmmCancel` and `cmmArchive` as if the branch was the main trunk. The only difference is that you must pass the branch identifier to `cmmModify`.

`cmmLast` lists all last archived versions for the trunk and for all *non closed* branches. A branch can be closed with the `cmmCloseBranch` command: the last version on the corresponding branch cannot be modified any more.

A module branch version can be retrieved with `cmmCopy` by passing the corresponding 4 digit version number.

Note that it is not possible to create branches on branches.

2.7 OTHER FEATURES

2.7.1 Knowing which module version you are using

To know which modules have been used to make a library or a binary file, use the `cmmWhat` command. `cmmWhat <file>` lists every CMM known module used in <file> with its version number.

```
$ cmmWhat readTest
read: 1.3+
ccs: 3.0
```

In this example, readTest is using read version 1.3 modified (1.3+) and ccs version 3.0.

2.7.2 Module history

`cmmHistory <module>` lists the history of the module: who has archived which version, when and why. `cmmHistory` simply records the `cmmArchive` command line.

```
$ cmmHistory read
version: 1.2 date: Mon Jan 23 10:53:03 MEZ 1995 user: vltsccm
> reason: creation
version: 1.3   date: Mon Jan 23 11:16:40 MEZ 1995   user: vltsccm
> reason: clean test
version: 1.4   date: Mon Jan 23 11:41:33 MEZ 1995   user: vltsccm
> reason: SPR 959999 & added readInt and readFloat
```

`cmmHistory` will also tells you if a module is currently modified or not.

2.7.3 Who is doing what

`cmmWho` lists the currently modified modules with check-out dates and user name.

```
$ cmmWho
<agws> modified by tphan since Mon Oct 14 18:47:17 MEZ 1996
<altaz> modified by bgilli since Mon Aug 19 15:07:10 MESZ 1996
<asmlcu> modified by ssandroc since Thu Sep 12 10:28:34 MESZ 1996
```

2.7.4 Last versions of module

`CmmLast <module>` gives the last archived version of the module and if currently modified, the user doing it:

```
$ cmmLast cmm
cmm              last version: 1.50     modified by vltsccm
```

Without any argument, `cmmLast` does the same for all existing modules.

2.7.5 Compare Module Versions

`cmmCompare` allows to compare two archived module versions.

```
$ cmmCompare read 1.3 1.2

Retrieving read 1.3. Please wait ...
Retrieving read 1.2. Please wait ...
DIRECTORIES:


FILES:
```

```
F:src/f.c ADDED
F:test/Makefile MODIFIED
==================================================================
diff test/Makefile 1.3 1.2
15a4
clean:   rm -rf *.diff *.err *.out
==================================================================
diff read/src/ReadString.c 1.3 1.2
11c11
< i=0;
---
> i=1;
                 < etc ... >
```

2.7.6 Retrieving an old version of a module.

`cmmCopy <module> <version>` retrieves a *read-only* copy of the module in the specified version. This is useful for integration purposes or to compare two module versions on a finer way.

2.7.7 Concurrency issues

CMM allows that the modification of same module version by two different users only with branching. But `cmmModify` locks the whole module: another user who want to modify the same version of the module with `cmmModify` can only wait for the user locking the module to archive it with `cmmArchive` or to cancel its modification with `cmmCancel` (in this case, all pending modifications are not archived). Furthermode, the following commands for a given module cannot run in parallel: `cmmCopy,cmmCompare,cmmModify,cmmArchive,cmmCancel`, `cmmArchiveBranch` or `cmmCancelBranch`.

The `cmmWho` command lists the modules currently under modification while the `cmmW` lists the ongoing `cmmCopy, cmmModify, cmmArchive, cmmCancel, cmmBranch, cmmArchive-branch` and `cmmCancelBranch` operations.

2.8 MODULE CREATION PREPARATION

The module creation is an operation reserved to the SCCM. But the software developer must prepare the work of the SCCM because he is the only one knowing the internal structure of his module. It is recommended to create a module as soon as possible, i.e. not to wait the last day before the delivery because the module preparation requires to test the module.

To prepare the module creation, execute the following steps:

2.8.1 module cleaning

First run `make clean`. Then clean your module using `cmmCheckForCreate:` this command has two steps:

1. checks the user write access on his files, tells to delete some backup files or some links
2. checks whether each file has the RCS keyword.

2.8.2 module backup

Save the module because the following step will modify your module and you must be able to retrieve a file before these modifications.

2.8.3 RCS keyword adding

Run `cmmAddRCSKeyword <mod>` on your module. This will modify any C source file by adding two static variable and any script by adding a commented line. This command will also add by default the standard script header to **ANY** other file (taking into account CCS command definition table file) where it is missing except those containing the line '*DO NOT EDIT MANUALLY THIS FILE*'.

Note that some files are not treated by `cmmAddRCSKeyword` because they don't use the standard templates. In this case add as a comment the following string "@(#) Id" (with the quotes) in the file header. Some files cannot contain yet the RCS keyword like the error file generated by the error editor or some test reference files or configuration files. In this case you have to **DELETE** what `cmmAddRCSKeyword` has generated and to restore the original file.

The best way to execute this step is to write a little shell script with the following actions:

```
cmmAddRCSKeyword <mod>
 <copy old files that cmmAddRCSKeyword should NOT have modified>
```

Loop over this step until all files displayed by cmmAddRCSKeyword *cannot* have the RCS keyword.

2.8.4 Test your module

Because almost each module file has been modified, automatically or manually, this is a mandatory step: compile and test your module.

2.8.5 Notify the SCCM.

Once your module is OK, do not modify it any more and get in touch with the SCCM: he will create the module for you. When the module has been created, you can modify it with `cmmModify` or get a read only copy with `cmmCopy`.

2.9 ADMINISTRATION COMMANDS

The following commands are reserved to the SCCM, the administrator of the code archive:

- `cmmInit`
- `cmmCreate`
- `cmmW`
- `cmmLockArchive`

```
-cmmUnlockArchive
-cmmStartup
-cmmShutdown
```

2.9.1 cmmInit

CAUTION !!!

This commands creates a new empty repository using the $CMM_ROOT environment variable as the path of the repository. It should be executed only once because it deletes the current repository. It only exists for repository creation and test purposes. As a consequence, this command is not installed under $VLTROOT and should be executed from the CMM module bin directory.

2.9.2 cmmCreate

This command allows the SCCM to create a module whose first version is by default 1.0.

2.9.3 cmmW

This command displays every ongoing CMM operation modifying the archive: `cmmCopy`, `cmmModify`, `cmmArchive`, `cmmCancel`, `cmmBranch`, `cmmArchiveBranch` and `cmmCancelBranch`. It is an easy way to check if an operation has failed and left a lock on the corresponding module by checking the starting date of the operation. Example:

```
$ cmmW
cmm     cmmModify by vltsccm since Thu Feb 27 10:33:44 MEZ 1997 on te13
```

2.9.4 cmmLockArchive and cmmUnlockArchive

`cmmLockArchive` allows to the SCCM to lock the archive, i.e. to disable every command for every user except the `cmmW` command (available for everyone) and the `cmmUnlockArchive` (only available for the SCCM). The parameter passed to `cmmLockArchive` is the message displayed by every command trying to access the archive. To unlock the archive, the SCCM must execute `cmmUnlockArchive`. Example:

```
$ cmmLockArchive “until 10:35”
Archive is locked.
$  cmmModify cmm
A R C H I V E  I S  L O C K E D : until 10:35
$ cmmUnlockArchive
Archive is unlocked.
```

2.9.5 cmmStartup and cmmShutdown

CMM working in client/server mode, the process `cmmServer` should be started on the server host to enable CMM. The `cmmStartup` commands takes care of this. `cmmStartup` can be executed interactively or at boot time: in both cases, it starts `cmmServer` iin background mode, using **$CMM_ROOT/cmm.log** as standard output and standard error.

`cmmShutdown` stops `cmmServer` after checking that no one is using the archive and after locking the archive.

2.10 New features

Starting with version 3.0, cmm has been reimplemented on a client server basis that doesn't use rcp any longer but a socket communication. Please refer to the [3] CMM Maintenance Manual Doc. No. VLT-MAN-ESO-17200-0793 for more specific information on the new architecture.
Briefly here we report the main changes that involve the users:

- to access cmm, once it is installed on your machine, every user must create a file

  $HOME/.cmmrc

  that must contain the record:

  username keyword

  The vltsccm is responsible for the creation of the keyword (just send a mail to vltsccm@eso.org). About the username, the default is: first letter of the first name + second name up to 7 letters (i.e. a username will be max. 8 character long)
- whenever a module is created, it is possible to declare it read-only or read-write for a certain group. Two main groups have been created on a project basis: VLT and ALMA; in ALMA another restricted group exists called ACS. To make an example, all the ALMA modules are read-only for all the users belonging to the ALMA group. Only a restricted group of users belongs to the ACS group and has write access also to the ALMA modules.

  For the moment, by default all the VLT modules are in r/w access for all the VLT users.

  This information is mainteined in 3 files:

  - modlist

    example:

    almatpl r=VLT: w=VLT:

    alrm r=VLT: w=VLT:

    altaz r=VLT: w=VLT:

    altaz.old r=VLT: w=VLT:

    altest r=VLT: w=VLT:

    ambcan r=ALMA: w=ACS:

    ambsi r=ALMA: w=ALMA:

    ampl r=VLT: w=VLT:

    amplx r=VLT: w=VLT:

    ...
  - userlist

    example:

    username1 keyword1

username2   keyword2

...

- grouplist

  example:

  username1 VLT

  username2 VLT

  username3 ALMA ACS VLT

  ...

Again, as the responsible person for the creation of a module is vltsccm, if you wish to have particular r/w permissions on your module, specify this in your request of creation to vltsccm@eso.org. By default a module belongs to the VLT group and will be r/w by all the users belonging to the VLT group.

3 Troubleshooting guide

3.1 Overview

CMM has been implemented as a client/server system: a CMM command uses the cmmClient program to send command to cmmServer. There is not a one to one mapping between a CMM command and the cmmClient calls: a single CMM command usually calls several times cmmClient.

cmmServer logs all incoming cmmClient requests and local CMM operation steps into **$CMM_ROOT/cmm.log**. This file is the only source of information to check wether a CMM operation has failed or not, even if the CMM command output is available on the client host. Do not delete this file, truncate it if it becomes too big and do not forget to back up it as every sub directory of **$CMM_ROOT**.

3.2 General failure

If any command fail with the following message: "`Cannot connect (CMM_HOST or cmmServer on CMM_HOST is down): Connection refused`", check the following:

- is it possible to reach from the local host the server host specified with $CMM_HOST ? run **ping $CMM_HOST** to check this. If this fail, there is a general network problem to be solved.

- if there is no network problem, check that cmmServer is running on $CMM_HOST by running **ps -eaf | grep cmmServer.** If cmmServer is not running, restart it as user **vltsccm** with the command **cmmStartup** (check **cmm.log** for an exit message of cmmServer that explains why cmmServer has exited).

It may also happen that cmmServer is running but that every command hangs: in this case, check that cmmServer is still alive with the basic **cmmW** command. If this command does not return, try to shutdown cmmServer and to restart it with the following commands as user **vltsccm**: **cmmShutdown**; **cmmStartup** ; if cmmShutdown also hangs, use **kill <cmmServer pid>** to kill cmmServer.

3.3 CMM operation failure

The CMM operations that modify a module archive have been designed to recover automatically. Before modifying the module in the archive, the module is backed up; if there is an error during execution, the operation fails but CMM restores the module and issues a error message.

3.3.1 A successfull CMM operation

Each successfull CMM operation may be decomposed form the server side into the following steps, that are also logged into **$CMM_ROOT/cmm.log**:

1. cmmLock <module>: this lock protects the tranfer of the module using **$CMM_ROOT/tmp** and the module update in the archive itself
2. cmmBackup <module>: backs up the module in the archive
3. cmm<Operation>: read and/or write the RCS files making up the module; the end is logged with "`cmm<Operation> <module> RCS op. end`"
4. the successfull final step of the CMM operation itself is logged with "`cmm<Operation> <module>  exit_OK`"
5. cmmUnlock <module>: the step is executed AFTER the module has been transfered on the client side, if needed.

A CMM operation is successfull IF AND ONLY IF all five steps above can be found in $CMM_ROOT/cmm.log, even if everything looks OK from the client side (no error message, module directory successfully retrieved).

3.3.2 A failed CMM operation with successfull recovery

Each unsuccessfull CMM operation that CMM manages to recover from may be decomposed from the server side into the following steps, that are also logged into **$CMM_ROOT/cmm.log**:

1. cmmLock <module>: this lock protects the tranfer of the module using **$CMM_ROOT/tmp** and the module update in the archive itself
2. cmmBackup <module>: backs up the module in the archive
3. cmm<Operation>: this failing step is logged with "`abort see:` **`$CMM_ROOT/tmp/log/<module>.<operation>.<pid>`**"
4. cmmRestore <module> is logged with "`cmmRestore <module> begin`" and "`cmmRestore <module> end`".
5. cmm<Operation> <module> is logged with "`exit_NOK (cmmRestore OK)`"
6. cmmUnlock <module>: this step is executed to enable other operations on the module.

This kind of failure may be caused by a temporary problem (file system full for example) or more likely by a CMM bug: only a carefull study of the corresponding file **$CMM_ROOT/tmp/log/<module>.<operation>.<pid>** may give further explanation.

3.3.3 A failed CMM operation with failed recovery

It is hard to give a general rule for such kind of errors because they may be caused by different reasons. Generally speaking, if running **cmmW** lists locks on old operations (more than 10 minutes after operation start), this means that something went wrong. A failure with no automatic recovery happens most of the time because a client or server process has been killed, possibly due to a network failure. It may also occur if the cmmRestore step failed. The only way to check that an CMM operation has failed this way is to check that **cmm.log** does not contain all the steps of one of the above two cases (successfull CMM operation and failed CMM with sucessfull recovery).

To restore the module in the archive and to enable operations on this module:

- for **cmmCopy** and **cmmBranch**: just run **cmmUnlock <module>**

- for **cmmModify, cmmArchive, cmmArchiveBranch, cmmCancel** and **cmmCancelBranch**: run:

$ cmmRestore <module>

$ cmmUnlockForFailure <module>

$ cmmUnlock <module>

Note that in the worst case (for example, running cmmRestore by hand has failed), the module should be restored from a disk backup. To restore module <*mod*>, you should restore three files ***and*** one complete directory (these three files and this directory should come from the same backup):

- **$CMM_ROOT/HISTORY/<mod>.history**
- **$CMM_ROOT/HISTORY/<mod>.fl,v**
- **$CMM_ROOT/HISTORY/<mod>.dl,v**

- **$CMM_ROOT/<mod>**

4 REFERENCE

This section provides a detailed description of all the CMM commands:

- cmmAddRCSKeyword
- cmmArchive
- cmmArchiveBranch
- cmmBranch
- cmmCancel
- cmmCancelbranch
- cmmCheckForArchive
- cmmCheckForCreate
- cmmCompare
- cmmCopy
- cmmHistory
- cmmLast
- cmmModify
- cmmWhat
- cmmWho
- cmmW

4.0.1 cmmAddRCSKeyword(1)

NAME

```
cmmAddRCSKeyword - adds RCS keyword to files in a directory tree
```

SYNOPSIS

```
cmmAddRCSKeyword [-n] <directory>
```

DESCRIPTION

```
Without -n option:

If the file is a C file:

- if the file contains the '#include vltPort.h directive the lines
  'char *rcsId="@(#) $Id...$";'  and
  'static void *use_rcsId = ((void)&use_rcsId,(void *) &rcsId);' are added.
- else if the file contains the 'VLT project' line, the line
'@(#) $Id...$" is added.
- otherwise, the file is not modified and listed as "not treated".

If the file is a script file (starting with !#) or a Makefile or
C include file the line "@(#) $Id...$" is added after the line
'VLT project'.

If the file is not a .c file and not a .cdt file that does not begin
with the standard header and if the file does not contain the line
"DO NOT MANUALLY EDIT THIS FILE", then the standard header is added
(with the RCS keyword: "@(#) $Id...$").

If the file already contains the RCS keyword, it is not modified.

With no option:

All rcsId definition in .c or .C files belonging to <mod>
are changed into:
static void *use_rcsId = ((void)&use_rcsId,(void *) &rcsId);

With -l option (for lcctest module only):
every file -except *.tar files- containing the string "SCCS Id ..."
is modified by replacing this string with "@(#)$Id: cmmAddRCSKeyword,v 1.69 1997/02/
24 13:42:26 vltsccm Exp $".
```

CAUTIONS

```
Assumes that files have been created with the getTemplate templates.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.2 cmmArchive(1)

NAME

```
       cmmArchive - checks in a module
```

SYNOPSIS

```
       cmmArchive <module_name> <reason> [<version>]
```

DESCRIPTION

```
       Registers the new version of the module.

       This command must be executed from the parent directory of the
       module.

       The <reason> argument should sum-up the modifications made on
       the module: typically, it's a SPR number and its short description.
       <reason> is a string that should be enclosed in quotes: this string
       is stored as a line in the module history file; this string can be
       arbitrary long but it is recommended that its length does not exceed
       80 characters.

       If no version number is given, the next module version is automatically
       computed by incrementing the last number (a.b will become a.(b+1)).
       If a valid version number is given, this version number will be the new
       module version (cmmArchive only checks that the new version is greater
       than the last archived version number (version gaps are possible).

       First, cmmCheckForArchive is executed. An error about read only files or
       files that should be deleted or links that should be deleted stops
       cmmArchive: the module is not archived and the module should be cleaned
       according to cmmCheckForArchive messages.

       Otherwise, each file, created, modified or deleted is taken into
       account: every existing module file, initially of version <a>.<b> will
       be registered with the new version number <a>.<b+1>: it is the new version
       of the module <module_name> (a deleted file can be recreated).

       Deleted and created directories are also taken into account.

       The current module directory is updated to become a read only copy of the
       module version <a>.<b+1> (files are read-only but directories are writable
       to be able to execute 'make').
```

FILES

```
       Requires:
       - repository, working directory

       Changes:
       - repository, working directory
```

RETURN VALUES

```
       0: module checked in.
       1: module not checked in but module archive OK.
       2: module not checked in AND module archive NOK.
```

CAUTIONS

```
       This script MUST NOT be interrupted. It may leave the repository
```

```
in a inconsistent state.

In case of failure and recovery failure, user must:
- ask 'vltsccm' to restore the module archive by hand
```

AND

```
- restore its own module with : tar xf $.mod.tar
```

SEE ALSO

```
cmmCheckForArchive, cmmModify, cmmCopy, cmmHistory.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.3 cmmArchiveBranch(1)

NAME

```
cmmArchiveBranch - checks in a module to create a branch
```

SYNOPSIS

```
cmmArchiveBranch <module_name> <reason>
```

DESCRIPTION

```
Registers the new version of the module by creating a branch:
the branch id has been allocated when running cmmBranch.

This command must be executed from the parent directory of the
module.

The <reason> argument should sum-up the modifications made on
the module: typically, it's a SPR number and its short description.
<reason> is a string that should be enclosed in quotes: this string
is stored as a line in the module history file; this string can be
arbitrary long but it is recommended that its length does not exceed
80 characters.

The new version of the module will be <branch id>.1.

First, cmmCheckForArchiveBranch is executed. An error about read only
files or files that should be deleted or links that should be deleted stops
cmmArchiveBranch : the module is not archived and the module should be
cleaned according to cmmCheckForArchiveBranch messages.

Otherwise, each file, created, modified or deleted is taken into
account: every existing module file, initially of version <a>.<b> will
be registered with the new version number <a>.<b>.<c>.1 (<a>.<b>.<z> being
the branch id allocated by cmmBranch): it is the new version
of the module <module_name> (a deleted file can be recreated).

Deleted and created directories are also taken into account.

The current module directory is updated to become a read only copy of the
module version <a>.<b>.<c>.1 (files are read-only but directories are
writable to be able to execute 'make').
```

FILES

```
Requires:
- repository, working directory

Changes:
- repository, working directory
```

RETURN VALUES

```
0: module checked in.
1: module not checked in but module archive OK.
2: module not checked in AND module archive NOK.
```

CAUTIONS

```
This script MUST NOT be interrupted. It may leave the repository
in a inconsistent state.
```

```
In case of failure and recovery failure, user must:
- ask 'vltsccm' to restore the module archive by hand
```

AND

```
- restore its own module with : tar xf $.mod.tar
```

SEE ALSO

```
cmmCheckForArchiveBranch, cmmBranch.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.4 cmmBranch(1)

NAME

```
cmmBranch - checks out a module to create a branch
```

SYNOPSIS

```
cmmBranch <module_name> <version>
```

DESCRIPTION

```
Creates a writable copy of the module <module_name> in the current
directory for modification purposes.

The archived version specified by the second argument is retrieved.

Once the writable module copy  has been created, files may be modified
added, deleted or re-created. Directories can be added, moved,
deleted or re-created.

If the retrieved version is <x>.<y>, the version number in the RCS
keyword is <x>.<y>.++ meaning that this is the version <x>.<y> currently
in modification to create branch <x>.<y>.<z>. This version number is
automatically updated in a string containing the '$Revision: 1.69 $' RCS
keyword after the module has been archived with cmmArchiveBranch or
retrieved via cmmCopy.
```

FILES

```
Requires:
- repository

Changes:
- working directory
```

RETURN VALUES

```
0: module checked out.
1: module not checked out but module archive OK.
2: module not checked out AND module archive NOK.
```

CAUTIONS

```
File executables permissions are not retrieved. The Makefile shall
take care of this (vltMake cares for standard file types: binaries,
scripts, etc.).

Do not use following names "quoted" with $ (i.e. $<name>$) in your files
because they will be modified by RCS : Author, Date, Header, Id, Locker,
Log, RCSfile, Revision, Source, State.
```

SEE ALSO

```
cmmArchiveBranch, cmmHistory, cmmCancelBranch.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.5 cmmCancel(1)

NAME

```
cmmCancel - cancel module checkout
```

SYNOPSIS

```
cmmCancel <module>
```

DESCRIPTION

```
Cancels the corresponding cmmModify <module> request:

- nothing is archived
- the user module directory is deleted.
```

FILES

```
Requires:
- repository, working module.

Changes:
- working directory.
```

RETURN VALUES

```
0: OK
1: module not 'cancelled' but module archive OK.
2: module not 'cancelled' AND module archive NOK.
```

CAUTIONS

```
Must be executed from the parent directory of the directory module
(i.e. cd <module_name> must be possible).

Does not warn about modified files which will not be archived:
cmmCheckForArchive may be used to get a overview of the pending
modifications but saving these modifications is left to the user.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.6 cmmCancelBranch(1)

NAME
```
       cmmCancelBranch - cancel cmmBranch
```

SYNOPSIS
```
       cmmCancelBranch <module>
```

DESCRIPTION
```
       Cancels the corresponding cmmBranch <module> request:

       - nothing is archived
       - the user module directory is deleted.
```

FILES
```
       Requires:
       - repository, working module.

       Changes:
       - working directory.
```

RETURN VALUES
```
       0: OK
       1: module not 'cancelled' but module archive OK.
```

CAUTIONS
```
       Must be executed from the parent directory of the directory module
       (i.e. cd <module_name> must be possible).

       Does not warn about modified files which will not be archived:
       cmmCheckForArchiveBranch may be used to get a overview of the pending
       modifications but saving these modifications is left to the user.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.7 cmmCheckForArchive(1)

NAME

```
       cmmCheckForArchive- checks module before archiving it it.
```

SYNOPSIS

```
       cmmCheckForArchive <module>
```

DESCRIPTION

```
       This command shall be executed from the parent directory
       of the module directory.

       Checks if the module exists and if the module has the minimum
       standard directory structure.

       Gives every directory added or deleted, and every modified,
       added or deleted file using cmmDiff.

       Gives the following error messages if:

       - files have unsufficient mode bits (-rw-r--r-- are the minimum set).
       - following file name patterns are present in the module directory:
         RCS, SCCS, *~, #*, .#*, *,v*, *.old, *.bak, *.BAK, *.orig, *.a , *.o,
         *.tar, *.Z, core, * * (file names containing blanks)
       - there are links in the module directory

       Warns about files that do not contain the RCS keyword, but
       does not check for binaries or man pages. ('make clean' should
       be run before cmmCheckForArchive).
```

FILES

```
       Requires:
       - repository, working module.
```

BUGS

```
       The '$Id' (end $ missing) RCS keyword for a file to be created
       is not catched.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.8 cmmCheckForArchiveBranch(1)

NAME

```
cmmCheckForArchiveBranch- checks module before creating a branch.
```

SYNOPSIS

```
cmmCheckForArchiveBranch <module>
```

DESCRIPTION

```
This command shall be executed from the parent directory
of the module directory.

Checks if the module exists and if the module has the minimum
standard directory structure.

Gives every directory added or deleted, and every modified,
added or deleted file using cmmDiff.

Gives the following error messages if:

- files have unsufficient mode bits (-rw-r--r-- are the minimum set).
- following file name patterns are present in the module directory:
  RCS, SCCS, *~, #*, .#*, *,v*, *.old, *.bak, *.BAK, *.orig, *.a , *.o,
  *.tar, *.Z, core, * * (file names containing blanks)
- there are links in the module directory

Warns about files that do not contain the RCS keyword, but
does not check for binaries or man pages. ('make clean' should
be run before cmmCheckForArchive).
```

FILES

```
Requires:
- repository, working module.
```

BUGS

```
The '$Id' (end $ missing) RCS keyword for a file to be created
is not catched.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.9 cmmCheckForCreate(1)

NAME

```
cmmCheckForCreate- checks module before creating it.
```

SYNOPSIS

```
cmmCheckForCreate <module>
```

DESCRIPTION

```
Checks if the module already exists and if the module has the minimum
standard directory structure.

Warns about files the owner cannot write.

Warns about the following file name patterns present in the module
directory that must be deleted:
Attic, RCS, SCCS, *~, #*, .#*, *,v*, *.old, *.bak, *.BAK, *.orig, *.a ,
*.o, *.tar, *.Z, core

Warns about links that must be deleted.

Warns about files that do not contain the RCS keyword, but
does not check for binaries or man pages. ('make clean' should
be run before cmmCheckForCreate).
```

FILES

```
Requires:
- working module
```

CAUTIONS

```
Must be executed from the parent directory of the directory module
(i.e. cd <module_name> must be possible).
```

BUGS

```
The '$Id' (end $ missing) RCS keyword for a file to be created
is not catched.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.10 cmmCompare(1)

NAME

```
cmmCompare - compare two module versions
```

SYNOPSIS

```
cmmCompare <module> <new version> <old version>
```

DESCRIPTION

```
First it gives the status of each file, with respect
to <new version> and <old version>:

- added
- deleted
- modified

Then, if the file is modified and is not binary,
the differences using 'diff' are also given.

Because of the module basis, each file existing in the two
versions are always considered as different because the RCS
keyword containing the version number is different.
```

FILES

```
Requires:
- repository
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.11 cmmCopy(1)

NAME

```
      cmmCopy - get a read-only copy of a module
```

SYNOPSIS

```
      cmmCopy <module_name> [ <version> ]
```

DESCRIPTION

```
      Creates a directory <module_name> in the current directory
      containing module <module_name> in version <version>.
      <version> by default is the latest one on the main trunk.

      Files cannot be modified: the module is given as read-only:
      but directories are writable so that 'make ' cab be executed.
```

FILES

```
      Requires:
      - repository

      Changes:
       - working directory
```

SEE ALSO

```
      cmmModify, cmmArchive, cmmHistory.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.12 cmmHistory(1)

NAME
```
cmmHistory - displays module history
```

SYNOPSIS
```
cmmHistory <module>
```

DESCRIPTION
```
Lists every archived version of the module with:

- version number,
- archiving date
- user who has performed this archiving
- comment (<reason> argument of the cmmArchive command).

If the module is currently modified with cmmModify/cmmBranch,  user(s)
modifying it are also listed.
```

FILES
```
Requires:
- repository
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.13 cmmLast(1)

NAME

```
cmmLast - displays module status (remote part)
```

SYNOPSIS

```
cmmLast <log_header> [ <module> ]
```

DESCRIPTION

```
Displays last archived version of <module> and if the module is
currently modified by cmmModify or cmmBranch, the user modifying it.
Last archived versions on branches are displayed if the corresponding
branch has not been closed with cmmCloseBranch.

If no argument is given, all modules status is displayed.
```

FILES

```
Requires:
- repository
```

RETURN VALUES

```
1 if <module> does not exist.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.14 cmmModify(1)

NAME

```
cmmModify - checks out a module
```

SYNOPSIS

```
cmmModify <module_name> [<branch_id>]
```

DESCRIPTION

```
Creates a writable copy of the module <module_name> in the current
directory for modification purposes.

If no branch id parameter is given, the last registered version
on the main trunk is retrieved. If a branch id parameter is given,
the last registered version on the given branch id is retrieved.
In both cases, if the module version is already under modification
(with cmmModify) by another user, no writable copy is produced. The lock
can be removed only by the user currently modifying the module, either by
archiving the modifications (with cmmArchive) or by removing the request
(with cmmCancel).

Once the writable module copy  has been created, files may be modified
added, deleted or re-created. Directories can be added, moved,
deleted or re-created.

If the retrieved version is <x>.<y>.[<z>.<a>], the version number in the
RCS keyword is <x>.<y>[<z>.<a>]+ meaning that this is the version <x>.<y>
[<z>.<a>] currently in modification. This version number is automatically
updated in a string containing the '$Revision: 1.69 $' RCS keyword after
the module has been archived with cmmArchive or retrieved via cmmCopy.
```

FILES

```
Requires:
- repository

Changes:
- working directory
```

ENVIRONMENT

```
CMM_ROOT    repository directory.
```

RETURN VALUES

```
0: module checked out.
1: module not checked out but module archive OK.
2: module not checked out AND module archive NOK.
```

CAUTIONS

```
File executables permissions are not retrieved. The Makefile shall
take care of this (vltMake cares for standard file types: binaries,
scripts, etc.).

Do not use following names "quoted" with $ (i.e. $<name>$) in your files
because they will be modified by RCS : Author, Date, Header, Id, Locker,
Log, RCSfile, Revision, Source, State.
```

SEE ALSO

```
      cmmArchive, cmmHistory, cmmCancel, cmmBranch, cmmArchiveBranch.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.15 cmmWhat(1)

NAME
```
       cmmWhat - give module names and versions
```

SYNOPSIS
```
       cmmWhat <file>
```

DESCRIPTION
```
       Gives module names and versions making up a file (binary or library)
       by searching <file> for the RCS keyword (@(#) $Id <..>$), sorting them
       and filtering them according to the module name.

       It assumes that every file belonging to the same module has the same
        version number.
```

FILES
```
       Requires:
       - repository
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.0.16 cmmWho(1)

NAME

```
cmmWho - displays which modules are being modified by whom
```

SYNOPSIS

```
cmmWho
```

DESCRIPTION

```
For every module checked out with cmmModify/cmmBranch commands,
the following line is printed:

<module> modified by <user> since <date>

If the module is checked out for branching (cmmBranch), the line
is completed with "for branch <bid>" where <bid> is the branch id
allocated by cmmBranch.
```

FILES

```
Requires:
- repository
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.1 SCCM commands

The following commands are reserved to the SCCM:

- cmmCreate
- cmmInit
- cmmLockArchive
- cmmShutdown
- cmmStartup
- cmmUnlockArchive

4.1.1 cmmInit(1)

NAME

```
       cmmInit - creates a new repository
```

SYNOPSIS

```
       cmmInit
```

DESCRIPTION

```
       This command can only be run by vltsccm.

       cmmInit creates a new repository which replaces the old one.
```

ENVIRONMENT

```
       CMM_ROOT    Repository path
```

CAUTIONS

```
       For the time being, there can be only ONE repository.
       Note that the old one is DELETED.

       THIS COMMAND DELETES THE CURRENT REPOSITORY.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.1.2 cmmCreate(1)

NAME

```
cmmCreate - create a module
```

SYNOPSIS

```
cmmCreate <module_name> [ <version> ]
```

DESCRIPTION

```
First, the directory is checked using the cmmCheckForCreate
command. If any error is detected, the module is not archived
and the module should be cleaned according to cmmCheckForCreate
prerequisites. If there is no error, every file under
<module_name> (including binary files) is registered and
given the version <version>.

By default, first version is 1.0.
```

FILES

```
Requires:
- working module

Changes:
- repository
- working module: all files are deleted.
```

CAUTIONS

```
This command is reserved to "vltsccm" user.

This command shall be executed from the parent directory
of the directory module (i.e. cd <module_name> must be possible).
```

SEE ALSO

```
cmmCheckForCreate
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.1.3 cmmLockArchive(1)

NAME

```
      cmmLockArchive - lock all archive
```

SYNOPSIS

```
      cmmLockArchive <message>
```

DESCRIPTION

```
      Disable every cmm operation, print <message>.

      This command can only be executed by "vltsccm".
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.1.4 cmmShutdown(1)

NAME

```
cmmShutdown - shutdown cmmServer
```

SYNOPSIS

```
cmmShutdown
```

DESCRIPTION

```
This command shutdown cmmServer after checking that no one
is updating the archive.

This command must be run on CMM_HOST.
```

ENVIRONMENT

```
CMM_ROOT should be defined.
```

SEE ALSO

```
cmmShutdown - cmmServer.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.1.5 cmmStartup(1)

NAME
```
      cmmStartup - start cmmServer.
```

SYNOPSIS
```
      cmmStartup
```

DESCRIPTION
```
      At boot time on HP-UX 9, use: /bin/su - vltsccm -c 'cmmStartup'
      (environments variables CMM_ROOT and CMM_PORT should be defined
       for user vltsccm).
```

ENVIRONMENT
```
      CMM_HOST, CMM_PORT and CMM_ROOT should be defined.
```

SEE ALSO
```
      cmmShutdown - cmmServer.
```

```
- - - - - -
Last change:  28/02/97-15:06
```

4.1.6 cmmUnlockArchive(1)

NAME

```
       cmmUnlockArchive - unlock all archive
```

SYNOPSIS

```
       cmmUnlockArchive
```

DESCRIPTION

```
       Undo cmmLockArchive: enable every end user command.

       This command can only be executed by "vltsccm".
```

SEE ALSO

```
       cmmUnlock
```

```
- - - - - -
Last change:  28/02/97-15:06
```

5 INSTALLATION GUIDE

5.1 GNU software installation on the server host

Installing CMM requires the following products to be installed on the **server** host:

- GNU diff 2.7
- GNU sed 2.05
- RCS 5.7

It is ***essential*** that the GNU diff, GNU sed, ci, co and rcs (RCS binaries) are the one above listed (otherwise CMM won't behave the expected way). Pay particular attention that **/usr/local/bin (**the installation target directory) precedes in the PATH environment variable a directory where diff, sed, ci or co utilities may already be installed. For example, on HP-UX 9.x, diff and sed are installed under **/bin** and RCS binaries are installed under **/usr/bin**.

5.1.1 GNU diff 2.7

To install:

```
$ gunzip diffutils-2.7.tar.gz
$ tar xf diffutils-2.7.tar
$ cd diffutils-2.7
$ ./configure --prefix=/usr/local
$ make clean
$ make
$ make install
```

Check that:

```
$ diff -v
diff - GNU diffutils version 2.7
```

5.1.2 GNU sed 2.05

To install:

```
$ gunzip sed-2.05-ss-9.01.tar.gz
$ tar xf sed-2.05-ss-9.01.tar
$ cd sed-2.05
$ ./configure --prefix=/usr/local
$ make clean
$ make
$ make install
```

Check that:

```
$ sed --version
GNU sed version 2.05
```

5.1.3 RCS 5.7

To install:

```
$ gunzip  rcs-5.7.tar.gz
$ tar xf rcs-5.7.tar
$ ./configure --prefix=/usr/local
$ make
$ make install
```

Check that:

```
$ ci -V
RCS version 5.7
$ co -V
RCS version 5.7
$ rcs -V
RCS version 5.7
```

5.2 Repository creation on the server host

- create a user account named `vltsccm` that will represent the owner of the repository:
- choose a directory name for the repository and assign it to the environment variable **CMM_ROOT.**
- as user `vltsccm`, run **cmmInit** from the **cmm/bin** directory module.

5.3 CMM module installation on the server host and on every client host

- run the following steps on the **server** host and on every **client** host

```
$ cd cmm/src
$ make clean
$ make
$ make man
$ make install
```

- choose a TCP port not yet used on the **server** host and assing it to the CMM_PORT on the **server** host and on every **client** hos*t*:

```
$ CMM_PORT=3000
$ export CMM_PORT
```

- define the environment variable CMM_HOST to the server host name for every **client** host:

  ```
  $ CMM_HOST=te13.hq.eso.org
  $ export CMM_HOST
  ```

  To use the server host as a client host, you should also do the same on the server host.
- Every CMM user should have a user account on CMM_HOST and should be able to execute a remote copy (`rcp`) from the **client** host to the **server** host *and* from the **server** host to the **client** host (see hosts.equiv man page for more information).

Check that running **cmmWho** does not return any error.

___oOo___