Software structure, management and distribution at KLOE
Draft 12/95 - V3.1
Kloe Memo N. 4

F. Donno

February 12, 1996

Contents

1 INTRODUCTION

The purpose of this document is to describe the directory structure and data and software organization at KLOE in the off-line environment for all platforms supported by the experiment (VAX/OpenVMS, DECAlpha/OpenVMS, HP/HP-UX, DECAlpha/OSF1), the version control system that can be adopted and the tools available to build and distribute off-site a given package or library. FORTRAN and C are both supported as high level languages. In the future also C++ will become a standard.

2 The Analysis Phase

Each user will be given a minimum of 10000 512-blocks in his/her own login area on each platform. This space can be increased up to 20000 blocks under request. In the login area and its subdirectories, the user develops and stores his/her own private code. Data and ntuples produced after running an analysis job reside in the official scratch areas, while data or ntuples of general interest go into official data areas (see next chapter). Private data can be backed up on tapes. For specific needs for online survival of specific data a generic KLOE data area will be created. In this area, after request, a subdirectory will be created with the same name as the requestor's username. This area cannot hold data for an infinite amount of time but for a certain period that the user has to specify while requesting the private data area. The same is true for software development. Private code, stored in the login areas, can become public after passing the approval of a library manager (see subsequent chapters). After this step, the code will become available to the entire collaboration and will reside in the public software areas.

3 The Data Organization

KLOE data will be available and accessible on all machines/platforms used by KLOE members inside the Frascati Lab. The data will be initially organized per detector. For each detector there will be directories for data coming from some analysis (filtering,etc.), from offline simulation, raw data, and ntuples. The role of the raw data directory can be redefined from time to time. Test beam raw data can be put there, as well as data coming from an official production job for the specific detector and ready for further analysis. In each directory we can find different subdirectories relative to the specific analysis or data type. The logical KD_XXX is defined to point to the XXX detector specific top data directory (see fig. 1). Not everybody will be able to write to these data areas but there will be a person responsible for each detector area. This person can request write access for his/her group of people. He/she will also be in charge of bookkeeping and cleaning tasks.

4 The Documents Organization

It is a general need to be able to access KLOE documents on-line on all machines/platforms used by KLOE members at Frascati Lab. All valid KLOE documents are in form of either either ASCII files with extension .TXT on OpenVms and .txt on UNIX, or valid TEX or LaTex documents with extensions respectively .TEX and .tex. Figures are postscript or GIF files with extension respectively .PS/.ps and .GIF/.gif. They can be eventually compressed using ZIP. There will be directories for public KLOE notes and publications, figures, KLOE packages manuals, KLOE notes. In each of these directories subdirectories can subdevide documents per argument or contain KLOE tex definition include file. Hopefully all documents will be available via WWW. The KLOE WWW sever will be configured in such a way that internal notes and information are protected. To access this information each KLOE collaboratore must have an account on the Frascati HP cluster. The logical/environmental variable KLOE_DOC will point to the documents top directory. The directory organization is as in fig.2.

5 Software organization: Utility packages and offline analysis libraries

A difference is made between utility packages and offline analysis libraries. A utility package could be either a Montecarlo package like GEANFI or a a product employed as basic tool for the physics analysis. Examples of the last mentioned case are YBOS, A_C, GEAPACK, etc.

We call a off-line analysis library a set of subroutines or functions that contain physics algorithms and that can be evenctually used to link a production executable. For instance, the subroutines or functions constituted by the A_C modules to reconstruct the tracks in the KLOE drift chamber are an example of the so called TRK offline analysis library.

Utility packages and offline analysis libraries have a different directory organization which is kept the same across OS platforms.

5.1 The directory structure for utility packages

A package code and object library are organized in a directory tree defined as follows:

Underneed the directory called with the same name as the package there is one subdirectory for each version of the package immediately available on disk. In the production subdirectory is always stored the most current working version of the package, while in development the version still under development can be found (see fig.3). Under the version subdirectory an optional OS flavor subdirectory exists. This is especially true for UNIX because of the etherogeneous nature of the cluster which shares common disks. In this case, specific libraries for each OS need to be stored in different places. This could also be the case for OpenVMS if at some point we have an etherogeneous cluster formed by old VAXes and new DECAlpha machines both running OpenVMS and sharing common disks. Under the version or flavor subdirectories there are the following subdirectories:

```
PROGRAMS/programs  where there are eventual binary
                   programs ready to use.

DOC/doc            for the package manuals that can also be
                   found in the public documentation area.

EXAMPLES/examples  examples area.

LIBRARY/library    home of include files,  debug  and
                   optimized  object libraries.
```

```
REFERENCE_COPY/    where the machine independent code is (see
local_copy         paragraph about how to write KLOE code).

SOURCE/source      where the machine dependent code is.

VPS/ups            where the setup  and unsetup procedures to
                   set up the package  logicals/environmental
                   variables take place (see the paragraph
                   "The KLOE Environment").

WORK/work          work area where temporary files are written.
```

Logicals/environmental variables to point to the subdirectories described above are set by the setup procedures and have the form of PACKAGE_XXX where XXX is the subdirectory name in capital (see the paragraph "The KLOE Environment").

5.2 The directory structure for an offline analysis library

The directory structure of an offline analysis library is much simpler (see fig.4)

Under the OFFLINE directory there is a subdirectory for each offline library. As for the packages also here we have version and optionally flavor subdirectories. Here, in the VPS/ups directories we can find the setup and unsetup procedures, while all code, including object libraries but EXCLUDING include files and machine independent source code, are in the same directory. One logical/environmental variable of the form K_XXX points to this area. The machine independent source code is kept in the area K_XXX/ref which is shared by the different flavor subtrees.

Include files of general use (i.e. used by almost all offline libraries) are in the area pointed to by K_INC. This area has the same structure as above but it has also one subdirectory for each offline analysis library containing all include files relative to that specific library, pointed to by the logical K_IXXX where XXX is always the name of the library.

6 The KLOE Environment

In order for a user to be able to set the KLOE environment at login time he/she has to have:

```
For OpenVMS, in his/her LOGIN.COM file the following line:

                $ @KLOE$COMMANDS:SETUP

For UNIX,    in his .cshrc file the following line:

 if ( -r ~kloe/group_cshrc ) source ~kloe/group_cshrc
```

```
and in his/her .login file the following line:

 if ( -r ~kloe/group_login ) source ~kloe/group_login
```

The directory pointed to by the logical/environmental variable KLOE_DIR contains a subdirectory for each KLOE package. The procedures above, beside defining some useful symbols for KLOE tools, also create the SETUP/setup symbol which allow users to set logicals/environmental variables to address the different packages of off-line libraries subdirectories. In what follows with the term "product" we will refer generically to a package or off-line analysis library.

When a user execute the command:

```
For OpenVMS:

        $ SETUP <PRODUCT> <VERSION> [ALL]

For UNIX:

        % setup [-e] <product> <version>
```

the setup/unsetup procedures in the VPS/ups directories of the respective product and version are searched and they are executed passing the arguments specified by the user on the command line. In the commands above the "ALL"/"-e" options are useful to execute the setup in an "extended" way, i.e. to setup ALL logicals/envs. which are used for instance by the librarian while building a library. The "ALL"/"-e" options are not useful to just use a library.

It is the task of the product maintainer to write adeguate and correct setup procedures for his/her own product. The SETUP.COM/UNSETUP.COM procedures for OpenVMS are generic DCL procedures, while on UNIX setup.csh/unsetup.csh are csh scripts. On UNIX, the command "man setup" gives a detailed description of the setup command and some hints about writing setup.csh/unsetup.csh scripts/ The procedures cannot hard-code disk or device names. They should run on all systems of the same type (i.e. not necessarily on Frascati machines but on all collaboration machines). Ask Flavia Donno (flavia@hpkloe01.lnf.infn.it) for specific needs or problems.

The offline itself as a whole could be considered as a package. The command:

```
For OpenVMS:

        $ SETUP OFFLINE <VERSION>

For UNIX:

        % setup offline <version>
```

calls the specific setups for all offline libraries for that version. The default version, if not specified, always points to the most recent working version of the offline when available. The offline setup provides also users with all logicals needed in order to link a production executable via the Link_Ana command (for a description see the A_C manual). It is recommanded not to put the "setup offline" command in the .cshrc file under UNIX. Infact the "setup" command sends on the screen certain messages and this fact makes the rsh/remsh command to fail. You are free to put the "setup offline" command in your .login file.

7 How to write in KLOE software and data areas

The public software and data areas have restricted access. In particular, for each software package or library there is a person responsible: the librarian. This librarian is the person responsible for the software in the production and development versions and consultant for eventual problems and bug fixes. To this person complains and suggestions should be sent for all pieces of a given package/library. The librarian also decides who are his/her collaborators. Collaborators can have write access to the library/package area.

The same happens for public data areas. Again, for each detector there is a group leader who is on charge for space management, bookkeeping and cleaning tasks. This person can request write access for the group members.

On VMS and UNIX systems, the librarian or group leader requires write access to a given software or data area for him/herself and the collaborators to the KLOE Off-line Computing Group by sending e-mail.

7.1 The write mechanism under UNIX

On UNIX machines, the write mechanism is based on the concept of a group. The data area directory belongs to a group named kloexxxd where xxx identifies the detector or physics area these data are related to. For the software area directories the name of the group assumes the form of kloexxx where xxx identifies again the detector or physics area, for offline libraries, while for packages the name of the group is equal to the name of the package. The following is the example of the trigger data area:

```
hpserver:~> ls -al $KD_TRG
total 24
drwxrwsrwx   6 kloe     kloetrgd     512 Feb 17 10:18 .
drwxr-xr-x   4 root     root         512 Feb 16 10:29 ..
drwxrwsr-x   2 kloe     kloetrgd     512 Feb  7 16:27 ana
drwxrwsr-x   2 kloe     kloetrgd     512 Feb  7 16:27 ntu
drwxrwsr-x   2 kloe     kloetrgd     512 Feb  7 16:27 raw
drwxrwsr-x   2 kloe     kloetrgd     512 Feb 17 11:09 sim
```

The group kloetrgd owns the directories ana, ntu, raw, sim, and, as we can see, anyone belonging to this group can write in these directories. To see who belongs to this group the command "libwho" comes to help:

```
hpserver:~> libwho trg
kloetrgd:*:233:kloe,anulli,bloise,bossi,dalessio,danielev,patera,
```

```
                edrago,schwick
kloetrg:*:234:kloe,bloise,bossi,patera
hpserver:~>
```

In the above example we see that the following people have write access to the trg data areas: anulli, bloise, bossi, dalessio, danielev, patera, edrago, schwick; while for the offline library trigger software area, only bloise, bossi, and patera have write access.

As you see, in each list the account kloe appears. This is the software manager account and only the Offline Computing Group people have access to it.

Once a user knows he/she has write access to a given area, he/she needs one more operation in order to be able to write. The UNIX command "newgrp" is used for so. The syntax is:

```
% newgrp <group name>
```

The command above forks (or spawns) a new shell (and the user's .cshrc procedure is executed) in which the user belongs to the new group as shown in the example below:

```
hpserver:~> id
uid=229(kloe) gid=204(kloe)
hpserver:~> newgrp kloetrgd
hpserver:~> id
uid=229(kloe) gid=233(kloetrgd)
hpserver:~> exit
exit
hpserver:~> id
uid=229(kloe) gid=204(kloe)
hpserver:~>
```

Once a user finishes writing in the group data or software area, he/she exists from the shell by means of the "exit" command.

It is important to allow people in the same group to be able to modify the files that a user creates. So the default protection for a newly created file should allow read and write access to the group. This is done for everybody: once a user issue the newgrp command automatically his/her own umask or default file protections are set to allow group write access.

To launch a NQS batch job that is able to write in one of the public group data area, one should submit a csh script on the example that follows:

```
#@$-s /bin/csh # Script shell
#@$-r example  # Request name
#@$-lt 3600    # Queue time limit
#@$-eo         # Merge error and standard output
```

```
#@$-me          # Send mail upon termination
#@$             # End of QSUB job description
#
newgrp <data group name> << EOF
setup offline
setenv K_PRODI /nfs/axpals/kloe02/prod/todo
setenv K_PRODO /nfs/axpals/kloe02/prod
#
link_ana sel0.exe sel0.opt no noztoy
sel0.exe <  sel0.uic >& sel0.out
EOF
#
#End of NQS job
```

For problems or if you need help, please contact F. Donno.

7.2 The write mechanism under VMS

The VMS write mechanism is based instead on rights identifiers and access control lists. On the public software or data areas directories Access Control Lists (ACLs) are enforced. Ex:

```
FLAVIA > DIR/SEC KD_TRG:[000000]ANA.DIR

Directory KD_TRG:[000000]

ANA.DIR;1            KLOETRGD             (RWE,RWE,RE,RE)
          (IDENTIFIER=[PHI,FLAVIA],ACCESS=READ+WRITE+EXECUTE+CONTROL)
          (IDENTIFIER=[PHI,FLAVIA],OPTIONS=DEFAULT,ACCESS=READ+WRITE+
          EXECUTE+DELETE+CONTROL)
          (IDENTIFIER=KLOETRGD,ACCESS=READ+WRITE+EXECUTE)
          (IDENTIFIER=KLOETRGD,OPTIONS=DEFAULT,ACCESS=READ+WRITE+
          EXECUTE+DELETE)
```

Rights identifiers are used as the UNIX groups described in the previous paragraph. The name convention is the same as described for UNIX. So, KLOEXXXD is the right identifier for the data areas while KLOEXXX is the one for the offline libraries software areas, where XXX is the name of the detector or physics. In order to write to a given area, users have only to verify that they hold the right identifier associated to that area. The following command allows users to do so:

```
FLAVIA > KLOE_RIGHTS/HOLD KLOETRGD
ANTON
BLOISE
```

```
BOSSI
FLAVIA
MAURA
ANULLI
SCIUBBA
DISTEFANO
BOLOGNESIV
DIBENEDETTO
EDRAGO
VETTORETTI
KLOEQUOTA
SCHWICK
DIFALCO
```

Given a username, it is also possible to list all rights identifiers held by that user:

```
FLAVIA > KLOE_RIGHTS/HELD KLOEQUOTA
KLOE$EXP
KLOETRKD
KLOECALD
KLOETRGD
```

KLOEQUOTA is the software manager account, as the account kloe for UNIX.

As you can notice the procedures are quite different now between VMS and UNIX. The Offline Computing group is working to make them similar.

8 How to write KLOE code

In what follows we will refer to the set of files with file types listed below as "source code". The only file types supported are the following with all their derivations:

```
.KLOE,.C,.COM,.CSH,.CIN,.DBT,.DBS,.DAB,.UI%,
.TXT,.TEX,.OPT,.MIN,.MAK,.MMS
```

The .KLOE/.kloe files are FORTRAN files wich contain EXPAND directives (see the EXPAND manual). The EXPAND is a home-made preprocessor which allows for include statements that use logicals or environmental variables in both FORTRAN and C languages. It also allows users to write code in a machine-independent fashion, putting machine-specific code between IF expand directive. It is the EXPAND task to relize which platform it is running on and extract the specific code from the .KLOE/.kloe files creating .FOR/.f platform dependent source code.

Analogously, .CIN/.cin files are include files that may contain machine specific code. In .CIN/.cin files the user can put FORTRAN or C specific code. It is the EXPAND task to produce then the equivalent .INC/.inc and .H/.h FORTRAN and C version of the include file.

A job can use internal ASCII files. This files are called .DBT/.dbt to be distinguished from the YBOS formatted internal binary files which have the extension .DAB/.dab.

Database files have the extension .DBS/.dbs.

.MIN, .MAK/.mak, and .MMS files are make-based description files. They are used by the automatic procedures to build a library (see next sections). .MIN files are used under VMS by REBUILD to build the library from the primary sources. They are produced by MIN_BUILD. The .MAK/.mak files are the dependencies files used by rebuild on UNIX. Right now they are also created by MIN_BUILD on VMS and copied by the librarian to the UNIX machine. min_build runs under VMS and UNIX. In particular, only the version that runs under UNIX has the new enhancements. This version is automatically invoked by rebuild.

For packages, all source code (see type list above) is put into the reference area pointed to by the logical/environmental variable ¡PACKAGE¿_REFERENCE_COPY. For offline libraries, the source code resides in the area pointed to by the logical/environmental variable K_XXX or in K_XXX/ref where XXX stands for the library name. So in this last case the area K_XXX/ref is the reference area for the library.

The FORTRAN or C Code in the form of machine-independent code in .KLOE or .C files should work on all platforms, OpenVMS and UNIX, supported by the experiment. The following rules should be respected.

8.1 How to write FORTRAN portable code

In this section we list the rules that should be followed in order to write FORTRAN portable code:

a) Always initialize variables before use. Integer variables can be initialized to 0, real to 0.0, characters to blank.

b) Never assume that variables are saved (which means that when you come back to a routine, local variables may have a different values than when you where in the routine the previous time). If you want to SAVE a variable you have to do it explicitely via the following statement:

```
SAVE <variable name>
```

c) You cannot open files that have an organization different than SEQUENTIAL (so, no indexed or direct organization is allowed). In the future UIPACK routines will be provided to deal with generic file I/O operations.

d) No logicals/environmental variables are allowed in OPEN or INQUIRE statements on UNIX. A mechanism is provided to use logicals/environmental variables:

```
$$IF UNIX
                INTEGER SITRFN
$$ENDIF
                .....
```

```
                .....

$$IF UNIX
                STATUS = SITRFN('MYLOGICAL',FULNAM,FULLEN)
                IF (STATUS .NE. SISUCC) UFLEN = 1
                OPEN(UNIT=LUN,FILE=FULNAM(1:FULLEN),
          &          ACCESS='SEQUENTIAL',
          &          STATUS='UNKNOWN',IOSTAT=STATUS)
$$ELSE
                .....
$$ENDIF
                .....
```

e) Avoid system calls. All system calls should be performed via calls to S_I (System_Interface) routines. If you do not find in S_I a routine to do what you want, ask the Offline Computing Group to provide you with one.

f) If you cannot avoid calling system specific routines, always use EXPAND directives to isolate the call for the specific system: "$$IF UNIX" for generic UNIX; "$$IF HPUX" for HP/HP-UX, "$$IF OSF" for DEC/OSF1, "$$IF VAX" for DEC/OpenVms.

g) Always allign common blocks by declaring first variables that occupie more bytes and than the ones that occupie less:

```
         COMMON/<common block name>/<double>,<integer>,<logical>,
     &                                <character>
```

Follows this rules also while declaring subroutines local variables.

h) If you are declaring a function or subroutine that takes no arguments, always use the open and close parenthesis after its name:

```
              INTEGER FUNCTION MYFUNCT()

              SUBROUTINE MYSUB()
```

i) Never call a function as a subroutine using the CALL statement. This causes a compiler error.

j) Always return from a function or subroutine call with a RETURN statement.

k) It is suggested that you create only one program section per file, i.e. one subroutine or function or block data or main program per file.

l) Do not put main programs in object libraries.

8.2 Building a library

To build a utility package or offline library, sitting in the reference area you have to issue the following commands (under UNIX, first copy all necessary source files in the package/offline reference area):

```
    For OpenVMS:

       For a Package:

        $ SET DEFAULT <PACKAGE>$REFERENCE_COPY
$ REBUILD <PACKAGE> <VERSION>

       If you want to run MIN_BUILD by hands:

$ MIN_BUILD/MACHINE=ALL/USE=KLOE <PACKAGE>$REFERENCE_COPY <PACKAGE>

       For an Offline library:

        $ SET DEFAULT K$XXX
$ REBUILD XXX <VERSION>

       If you want to run MIN_BUILD by hands:

        $ MIN_BUILD/MACHINE=ALL/USE=KLOE K$XXX XXX

     For UNIX:

       For a Package:

        % cd $<PACKAGE>_LOCAL
% rebuild <package> <version> nodebug
% rebuild <package> <version> debug
% rebuild <package> <version> O2
% rebuild <package> <version> .....

       If you want to run min_build by hands:

        % cd $<PACKAGE>_WORK
% min_build -machine=all-use=kloe <PACKAGE>_LOCAL <package>

       For an Offline library:

% rebuild <package> <version> nodebug
% rebuild <package> <version> debug
% rebuild <package> <version> O2
% rebuild <package> <version> .....
```

```
If you want to run min_build by hands:

 % cd $K_XXX
% min_build -machine=all-use=kloe K_XXX xxx
```

Note that while under VMS the rebuild command automatically generates the DEBUG and NODEBUG/OPTIMIZED version of the library, under UNIX the explicit request must be specified on the rebuild command line. This is done to allow the librarian to choose what optimized version of the library he/she wants to serve publicly on disk, and so to save disk space. If the code complies to the rules specified in this document and in the KLOE Software Guidelines, you should be able to get a library.

8.3 Invoking the EXPAND and FORTRAN compiler under UNIX

The expand and fortran commands are quite different under UNIX than under VMS. All expand parameters and qualifiers that a user provides under VMS can still be specified under a UNIX but with a slightly different syntax: all qualifiers are preceeded by a "-" sign instead of the "/" sign.

```
On HP-UX you expand a .kloe file with:

   % expand -env=hpux-option=FORTRAN-comment=SOURCE myfile.kloe

   or simply:

   % expand -option=FORTRAN-comment=SOURCE myfile.kloe

On DEC/OSF1 you expand a .kloe file with:

    % expand -env=osf-option=FORTRAN-comment=SOURCE myfile.kloe

    or simply:

    % expand -option=FORTRAN-comment=SOURCE myfile.kloe
```

Be careful, the commands above are case sensitive. The output of the expand command is a .f file. The fortran compiler is invoked in a different manner under HP-UX and DEC/OSF1. The UNIX aliases for and ford are provided to produce a .o object file respectively in nodebug and debug mode under the two platforms.

```
Under HP-UX:

    % fort77 +ppu -c [-g] myfile.f

Under DEC/OSF1:
```

```
% f77 -c -G 0 [-g] myfile.f
```

```
If you use the for/ford aliases, on both platforms:
```

```
% for[d] myfile.f
```

The "-g" option is used to produce a debug version of the object file. If your only goal is to rebuild a library, you do not need to use the above commands at all since rebuild will use them for you. If you are building a job, then you need to expand and compile all your modules and the build_job output. The link_ana option file under UNIX should be a list of .f or .o files and .olb/.dlb libraries ordered in one column and separated by nothing (no commas or VMS continuation dashes).

```
% more myopt.opt
mybuild_job.o
mymodule.f
$K_TRK/trklib.dlb
$K_EMC/emclib.olb
```

9 The code distribution mechanism

The code distribution mechanism is still under development. While under OpenVMS we have a working version, the UNIX part is not complete. Both mechanism are basically based on the REBUILD/rebuild procedures. The idea is that the source code in the reference copy areas of each package and offline library is copied at the remote sites. There the rebuild procedure is run to build the object libraries. Before starting, you should get a backup/tar tape with the basic KLOE installation. This tape will contain the first installation toolkit together with the directory structure as described above.

9.1 The OpenVMS code distribution mechanism

In the KLOE_DIR directory you can find two procedures called respectively UPDATE_CODE.COM and UPDATE_OFFLINE.COM. The UPDATE_CODE.COM is a procedure that can be run once a week to keep the KLOE packages up to date. It assumes that the source reference machine is AXKLO1, that the node AXKLO1 is known at the remote site and that there is a DECNET connection between AXKLO1 and the remote machine. Inside UPDATE_CODE.COM you can find a SUBMIT line which is the one responsible for every night/week job submission. Please, change this line to specify the batch queue you want the update batch to run and the name of the log file. By default the job is resubmitted every night. You can of course change this. UPDATE_CODE.COM should run before UPDATE_OFFLINE.COM is run. UPDATE_OFFLINE keeps the offline development libraries in sync with what is available on AXKLO1. The structure of this procedure is the same as for UPDATE_CODE.COM. You can customize it as you like.

9.2 The UNIX code distribution mechanism

The UNIX code distribution mechanism is quite uncomplete at the moment. Let's assume that you have all basics utilities installed at your site (ups - Unix Product Support, rebuild, directory structure, etc.). You can then update each package and offline library reference copy source code starting from what you find on hpserver at Frascati and then you can run rebuild locally to produce a library. Please, ask Flavia for more details.