E391-library manual
— Software kit for the E391a experiment —

M. Yamaga*

High Energy Accelerator Research Organization, KEK,

1-1 Oho, Tsukuba, Ibaraki 305-0801, Japan

March 11, 2003

Abstract

This document describes the software toolkit "E391-library" for the KEK PS-E391a experiment. The installation procedure, structure of the library, execution and code-development procedure are described in detail.

*electric address : yamaga@post.kek.jp

Contents

1 Introduction

The "E391-library" is originally developed to be a common software-tools for the KEK PS-E391a experiment. The aim of this library is to collect as much information necessary for this experiment as possible into one place, and make collaborators easy to access them. Using the common program library will also be useful to avoid the redundant duplication work among the collaborators.

The library is designed to work on the PC-linux, IBM-AIX and Sun-Solaris operation system that will be our main platforms. It supports the important feature, the "HPSS" — High Performance Storage System —, which is the large tape-storage system at KEK central computing system.

The distribution-package of the library is prepared so that one can obtain and install it into one's PC. It covers the software from frontend of DAQ to the utilities for the reconstruction and analysis in the offline stage.

The code-development is done by creating a small unit of directory-tree, so-called "package", into the users working area. The template of the package is available for the convenience of the quick development. The shared-library system will contribute to this style of development much easier and faster, because the developer can only copy and modify the code of interest, and that the updated code can be linked with the other part of the E391-library.

The version-control of each source file is done by CVS (concurrent versions system). The new version of the library will usually be released when the serious bug was fixed or the important package or the new feature was added into the library.

All the collaborators are encouraged to contribute to the development of this library. For the latest information of the library, visit the home page.

http://psux1.kek.jp/~e391/software

2 Installation

On the KEK central-computing system "ps.cc.kek.jp", the latest version is available under /dfs/g/ps/klea/e391/e391/pro , thus you can immediately use the feature of the E391-library. Also, most of the Linux-PCs of the E391a group,"kl-xx", and our DAQ computers, already have E391-library in it.

On the other computers, if the E391-library is not installed in it, you need to install it.

2.1 Requirement

The following utilities are necessary to install the E391-library.

- CERN library
- CLHEP library
  http://wwwinfo.cern.ch/asd/lhc++/clhep/index.html
  The current version is clhep-1.7.0.0.tar.gz.
- zlib ($\geq$1.1.4 recommended)
- gzip
- GNU tar
- GNU make (gmake)
- Perl ($\geq$5.004? $\geq$5.6 recommended)
- tcsh

2.2 Installation on PC linux

Step 1

Obtain the E391-library distribution-package from our home page.

```
http://psux1.kek.jp/~e391/software
```

The filename will be e391-exxxx_xxxx.tar.gz for source file and database-yyyy_yyyy.tar.gz for database file, where xxxx_xxxx and yyyy_yyyy mean the timestamp (year, date and time, e.g. 20030126_2336) that the tarball is created on. The timestamp for the source file and database are not necessarily be the same.

Step 2

Create the directory tree where the e391a library will be installed. /e391 is recommended for the top directory if possible, otherwise <dir>/e391 where <dir> stands for your working directory. Extract the package there.

```
% mkdir <dir>/e391
% mkdir -p <dir>/e391/e391/exxxx_xxxx
```

```
% cd <dir>/e391/e391/exxxx_xxxx
% tar xvzf e391-exxxx_xxxx.tar.gz
```

xxxx_xxxx should be taken from the package name. The examples and src directories will be created under the exxxx_xxxx directory.
If you do not have the database yet, extract it.

```
% cd <dir>/e391
% tar xvzf database-yyyy_yyyy.tar.gz
```

Step 3

Set the following environment-variables properly.

E391_HOME	<dir>/e391
E391_LEVEL	exxxx_xxxx
E391_TOP_DIR	${E391_HOME}/e391/${E391_LEVEL}
E391_LIB_DIR	${E391_TOP_DIR}/lib
E391_CONFIG_DIR	${E391_TOP_DIR}/config
E391_DB_DIR	${E391_HOME}/database
CERN	Top directory of CERN library (e.g. /cern)
CERN_LEVEL	(e.g. 2000)
CERN_ROOT	${CERN}/${CERN_LEVEL}
CLHEP_BASE_DIR	Top directory of CLHEP library (e.g. /usr/local)
CLHEP_LIB	CLHEP library file excluding the suffix .a (e.g. CLHEP-g++.1.7.0.0)
LD_LIBRARY_PATH	.:${E391_LIB_DIR}/so:${E391_LIB_DIR}:${LD_LIBRARY_PATH}

The sample script to set these variables is prepared in the E391-library package for your convenience.

```
examples/scripts/cshrc_e391
examples/scripts/bashrc_e391
```

It is recommended to copy them to <dir>/e391/local/etc/cshrc_e391(bashrc_e391) so that everyone can easily set up these variables correctly. Modify it to match your computer's environment, then source it.

```
% source <dir>/e391/local/etc/cshrc_e391   (for csh, tcsh)
% source <dir>/e391/local/etc/bashrc_e391   (for sh, bash)
```

Step 4

Build the e391 library.

```
% cd <dir>/e391/e391/exxxx_xxxx/src
% gmake
```

This will do all necessary thing to build the utility libraries and executables.

After finishing the build, new directories `<dir>/e391/exxxx_xxxx/{bin,include,lib,share}` will be created and many files will be there.
If you do not need to keep the intermediate files as `.o`,

```
% gmake clean
```

will erase the intermediate files in the library.
After the installation, the directory-tree will be something like Figure 1.

3 Execute the program

E391-library contains many useful programs. One can simply use these programs as a tool.

3.1 Setup

One must setup some environmental variables in order to run the executables in the E391-library.

```
PATH             ${E391_TOP_DIR}/bin:${E391_HOME}/local/bin:${PATH}
LD_LIBRARY_PATH  .:${E391_LIB_DIR}/so:${E391_LIB_DIR}:${LD_LIBRARY_PATH}
```

`LD_LIBRARY_PATH` is not necessary on ps.cc.kek.jp since the shared-object feature is disabled on ps.cc.kek.jp due to the restriction from the HPSS library.

These setup can be done by the script `<dir>/e391/local/etc/cshrc_e391`, so it is recommended to source `cshrc_e391(bashrc_e391)` in your login script, `${HOME}/.cshrc` or `${HOME}/.bashrc`.

3.2 Run

Let's run the event display program as an example. The executable `eventDisplay` should be installed at `${E391_TOP_DIR}/bin/eventDisplay`. For example on the Linux-PC `kl-3`,

```
% which eventDisplay
/e391/e391/pro/bin/eventDisplay
```

The `eventDisplay` has many input parameters.

```
% eventDisplay
usage : eventDisplay [options]
  options:
    -h <hostname> (connect to the running experiment.)
    -i <filenames...> (you can use %05d for run_number. e.g. run%05d.mid)
    -r <run#from> <run#to> (work with %05d)
    -n <#event>
    -skip <skipevent>
```

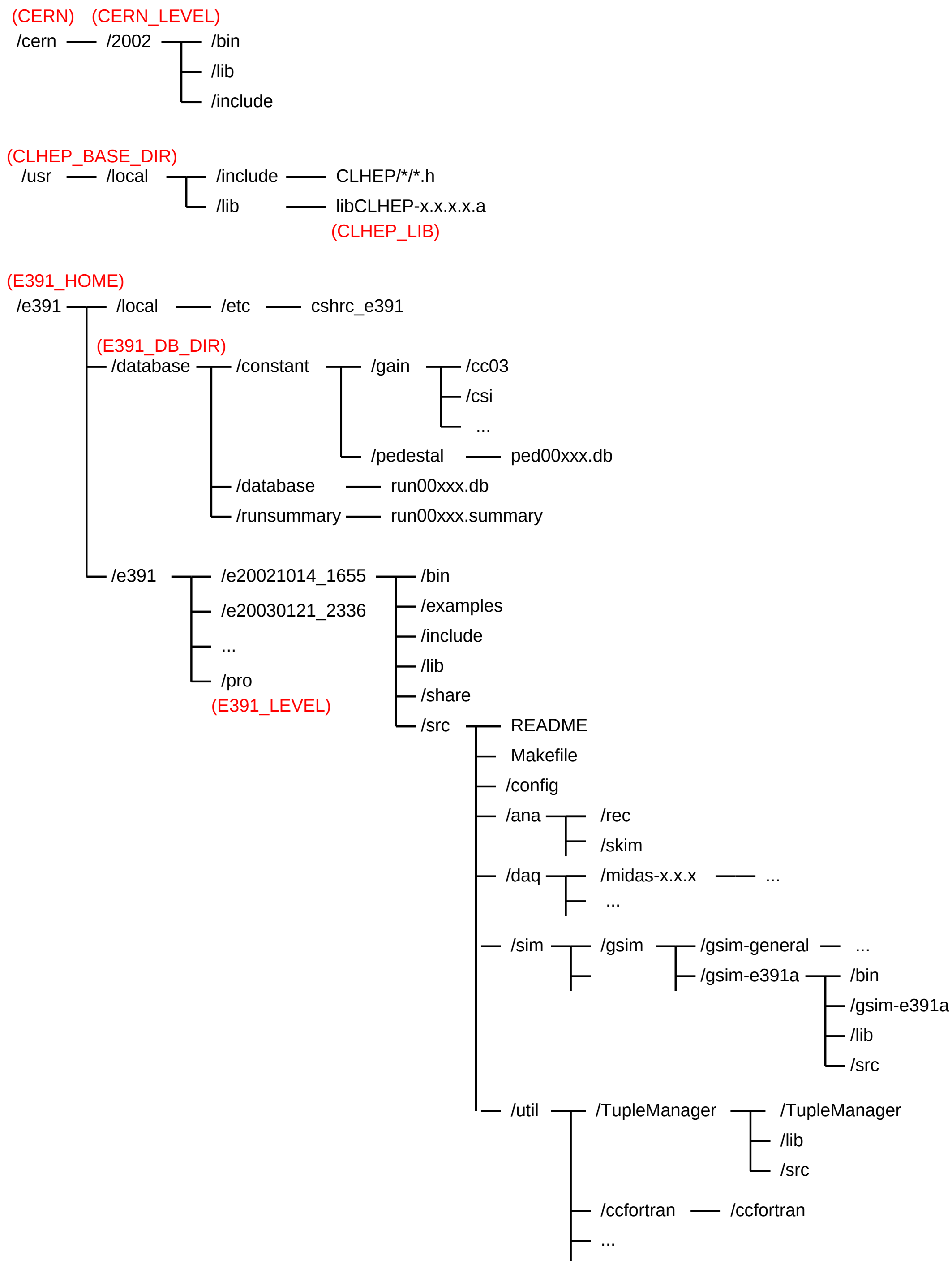

Figure 1: The directory-tree of E391-library.

```
-adccut <low> <high> (ADC cut-off: low-range high-range)
-nsigma <n> (ADC cut-off < mean + n*sigma)
-int <n> (display once per n_event)
-pause (pause until hitting 'enter' key)
-cycle <n> (n second cycle of beam extraction, FOR ONLINE ONLY)
-freq <n> (display n event per sec, FOR ONLINE ONLY)
-usleep <n> (usleep n usec)
-trig <n> (trigger bit)
-rot <n> (rotation angle)
-tracking (tracking the cosmic ray)
-psfile <filename>
-noscreen (Do not display on the screen. Usable with -psfile option.)
```

Let's view the event of run #1030 on `kl-3`. The data file is located at
`/work2/e391data/build/build01031.mid`
The following command will show the first event and make pause. You will see the picture like Figure 2. Hitting the 'enter' key will process the next event one by one. Hitting 'Ctrl-C' will terminate the program.

```
% eventDisplay -i /work2/e391data/build/build01031.mid -pause
```

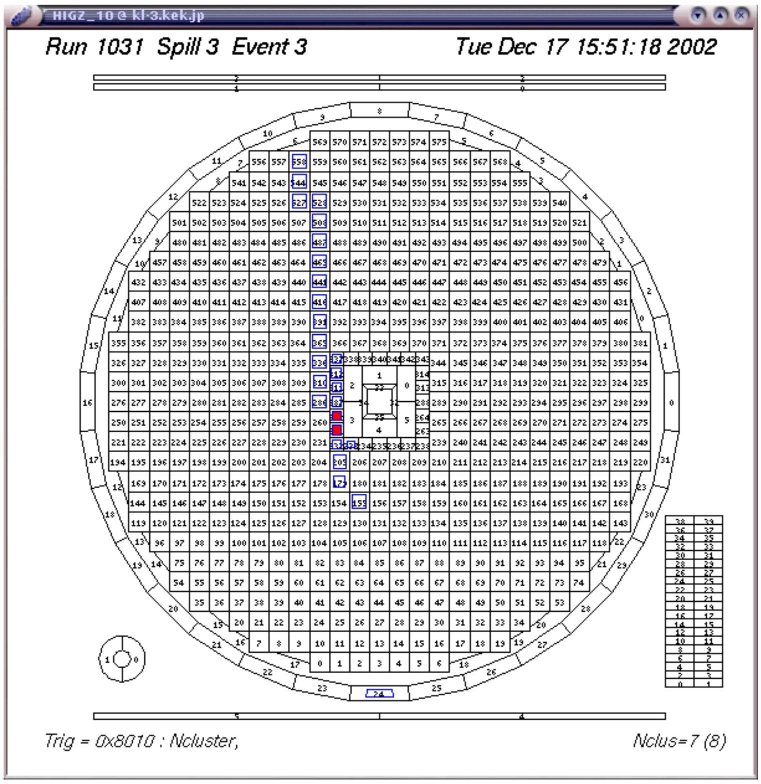

Figure 2: The sample event display.

4 The code development with the E391-library

You may need to write a program for your analysis. Sometimes you may need to modify the present source code for the update of the library. You may also want your code to be included in the library so that other collaborators can use the useful routine that you developed.

The E391-library has the convenient system for those purpose. The "package" system, which is the unit of the program code with the small directory-tree, will offer the quick and easy way for the code development.

4.1 Package system

Figure 3 shows the unit of "package". Source code should be located in the `package-name/src` directory. The header-file, include-files should be located in the `package-name/package-name` directory.

The template of the package `skeleton` is included in the `example/` directory.

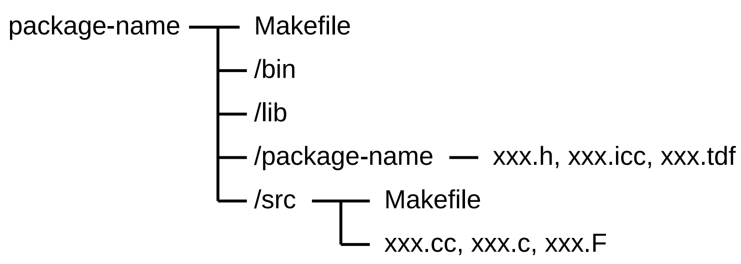

Figure 3: The unit of the code development "package".

In principle, all the include-files as `.h`,`.icc` and `.inc` in the `package-name/package-name` will be copied (actually symbolic-linked) into the `e391/include/` directory as is written in the next section. All the C++, C and FORTRAN source file in the `package-name/src` directory will be compiled and linked as a library or an executables according to the description in the Makefiles.

4.2 `get_packages` command

Since the E391-library is fully managed by the package system, one also need to use the package system for the development. In order to modify the present code in the E391-library, one must create the branch of the directory-tree. The branch has the similar structure as the E391-library tree, as shown in Figure 4. All the work should be done inside the branch.

One must set the special environmental variables.

```
MY_TOP_DIR       your_local_dir/e391
MY_LIB_DIR       ${MY_TOP_DIR}/lib
MY_CONFIG_DIR    ${MY_TOP_DIR}/config
LD_LIBRARY_PATH  .:${MY_LIB_DIR}/so:${MY_LIB_DIR}:${LD_LIBRARY_PATH}
```

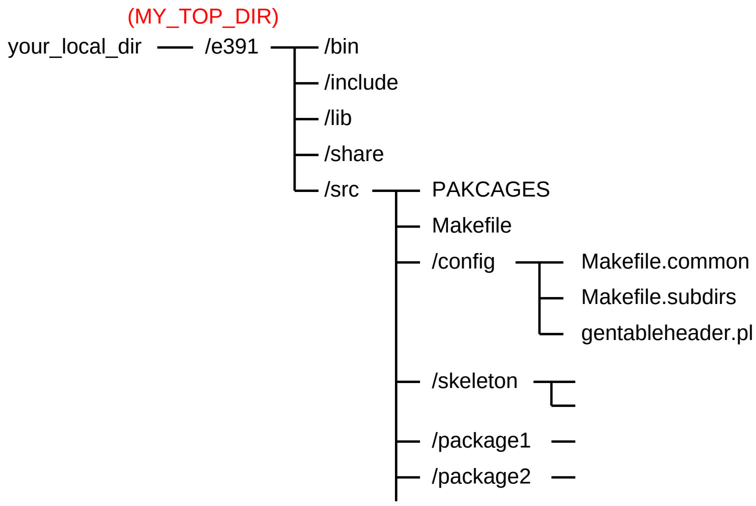

Figure 4: The local directory tree for the development.

These variables will be set correctly by the script `<dir>/e391/local/etc/cshrc_e391` only if you set the `MY_TOP_DIR` in advance by yourself.

```
% setenv MY_TOP_DIR your_local_dir/e391
% source <dir>/e391/local/etc/cshrc_e391
(% source <dir>/e391/local/etc/bashrc_e391 for the bash)
```

After setting the environmental variables, one must create `e391/src` directory. Move to there, and issue the `get_packages` command which is installed in the `${E391_TOP_DIR}/bin/`. This command creates the tree structure of the branch, copies the packages and modify some part of the description in the Makefiles to cope with your local environment. The local directory-tree will be as Figure 4.

```
% mkdir -p your_local_dir/e391/src
% cd your_local_dir/e391/src
% get_packages skeleton
```

The main difference between the official library-tree and the local tree is that all the packages will be located under the `e391/src` directory, and do not form the further hierarchical structure.

The `get_packages` command accept the multiple package at a time. One can also later append the package to the local directory.

```
% get_packages ccfortran TupleManager
% get_packages gsim-e391a
```

Above command will copy the ccfortran, TupleManager and gsim-e391a packages in your local directory. After issuing the `get_packages` command, you should build the packages first by `gmake` in the `e391/src` directory in order to check whether your local environment is correctly work or not. After making sure it works, you may modify the source code.

4.3 The priority of the linkage of the libraries

When you build your local package, the updated include-files and updated libraries are installed in `your_local_dir/e391/(include,lib)`, respectively. These local files have higher priority to the official location in the description of the local Makefiles for the compilation and linkage process. Thus you can always use your local files prior to the official one only if you correctly set the `MY_XXX` environmental variables.

4.4 Packages to get

Suppose that you want to modify the detector geometry of the `gsim` (GEANT3 Monte Carlo simulator). The `gsim` is included in the `gsim-e391a` package. In this case, getting only the `gsim-e391a` package will be enough.

Another case, suppose that you want to modify the E391Detector class to add the new feature, and want to test the gsim with the updated E391Detector class. In this case, getting only the `E391Detector` package might not sufficient. You should also get `gsim-e391a` which contains the executables you want to test (i.e. `gsim`).

4.5 Creating the new package

When you want to create the new package, the easy way is to copy the present package and rename it. The `skeleton` package will be good for this purpose.

```
% get_packages skeleton
% cp -r skeleton foopackage
% mv foopackage/skeleton foopackage/foopackage
```

The package name must be appear in the text file `e391/src/PACKAGES` in order to be processed by `gmake`. `get_packages` command automatically append the obtained module name in the `e391/src/PACKAGES` except for `skeleton`. In the case that you created the new packages according to the procedure above, you must append the name of the new packages by hand.

```
% echo foopackage >> PACKAGES
```

4.6 Modification of the Makefile

There are two Makefiles in a package : `foopackage/Makefile` and `foopackage/src/Makefile`. In most of the case, you do not need to modify the first one, `foopackage/Makefile`.

```
# -*- Makefile -*-

INSTALL_BINS = yes
INSTALL_LIBS = yes

SUBDIRS = src
```

```
 7
 8  ### General rules; DO NOT EDIT.
 9  PACKAGE = $(shell basename `pwd`)
10
11  packagedir = .
12
13  include $(MY_CONFIG_DIR)/Makefile.subdirs
```

The line#3 and #4 can control whether the executables and libraries in this package will be installed in the `your_local_dir/e391/(include,lib)` directory. Specifying `yes` will install them, and `no` will not. Only for the special purpose you can use them.

The second one, `foopackage/src/Makefile`, has important descriptions for the development.

```
 1  # -*- Makefile -*-
 2
 3  MAINS =
 4
 5  # For library
 6  LINK_ARGS_SO = -L$(MY_LIB_DIR)/so -L$(MY_LIB_DIR)
    -L$(E391_LIB_DIR)/so -L$(E391_LIB_DIR)
 7
 8  # For executable
 9  LINK_ARGS_BIN = -L$(MY_LIB_DIR)/so -L$(MY_LIB_DIR)
    -L$(E391_LIB_DIR)/so -L$(E391_LIB_DIR) -l$(PACKAGE) -lTupleManager
     $(CLHEPLIB) -L$(CERNLIB_DIR) -lpacklib -lmathlib $(SYSLIB)
10
11  ### General rules; DO NOT EDIT.
12  PACKAGE = $(shell basename `cd .. && pwd`)
13
14  packagedir = ..
15
16  srcdir = .
17
18  include $(MY_CONFIG_DIR)/Makefile.common
19
```

In the line#3 (`MAINS =`), the list of the name of the executable should appear. The name is the same as the source file excluding the suffix (.cc/.c/.F). For example, if you want to create two executables, myana1 and myana2, the filename myana1.cc(or .c,.F) and myana2.cc(or .c,.F) must exist in the `foopackage/src` directory, and the line#3 should be as follows.

```
MAINS = myana1 myana2
```

Line #6 is used only if you build the library. You should append here the additional library name to be linked with the shared-library. In most of the case you do not need to add any library name here. The name of the library to be produced in this package is the same as the name of the package as specified in the line #12. For example, the library name in the `foopackage` will be `libfoopackage.(a/so)`. Only one library per package is available. If you do not want to create the library, comment out the line #12 and #6, even though line #11 says that you must not edit line #12.

```
#PACKAGE = $(shell basename `cd .. && pwd`)
```

Line #9 is used only if you build the executables. Comment it out if you do not build the executables. You should append/remove here the additional library name to be linked with the shared-library. In most of the case you have to modify this line so that the loader correctly resolve all the symbols required. If you build more than one executables, and if the required libraries are different between them, you can specify the required libraries for each executables by the description as follows.

```
myana1:-lmathlib myana2:-lgeant321 myana2:-ldl
```

If you want to give some options to the compiler, you can describe `CXXFLAGS`,`CFLAGS` or `FFLAGS` as follows.

```
CXXFLAGS += -Wall
```

Same description is applicable for the loader option.

```
LDFLAGS += -Wall
```

These description should appear before the line #18. Line #4 will be the convenient position for these descriptions to appear.

Some of the standard libraries are defined as variables like `SYSLIB` in the Makefiles in the official directory, `$(E391_TOP_DIR)/src/config/Makefile.xxx`. If you met some problem about the Makefile, consult the Makefiles in the other packages in the official library, and you will obtain many hints from them.

5 Version control by CVS

5.1 Update and download

The version-control of each source file is done by CVS (concurrent versions system). Only the person who has the permission to access the CVS-repository can directly update and download the official-release code. Since many modification from many people at a time might break the consistency between the modifications, this restriction is necessary to keep the quality and the stability of the official-release. Those who cannot have the permission to check-in have to ask someone who has the permission to check-in their code.

The new version of the library will usually be released when the bug was fixed or the new feature was added into the library. The frequency of the new-release will vary time to time. The announcement of the new-release will be circulated by e-mail to the collaborators. Use only the version that is announced as a stable release.

Although the direct access to the CVS repository is restricted, one can obtain each file by the CVSweb tool. CVSweb is now working on `kl-7`, so you can check the latest status of each file of the E391-library in the CVS repository, and even obtain each file. If you want to try the latest updates even if it is not announced as stable, check the web site below.

`http://kl-7/cgi-bin/cvsweb.cgi/`

5.2 Set-up of the CVS specific to the E391-library

In order to manage the package-system of E391-library, some settings are required to the CVS.

Each package should be checked out by its package-name. Thus CVS need to have the reference table of the package-name to the actual directory position in the CVS repository. `CVSROOT/modules` is the file for the reference table. Accordingly, the restriction appears that the same package-name in different directory is not allowed in the E391-library. Each package should have the unique name.

When the check-in of the updates was done, the log-message is accumulated into the `CVSROOT/commitlog` file. After the check-in of the updates, e-mail will be sent to the responsible people. These settings are described in the `CVSROOT/loginfo` file.

6 Trouble shooting

6.1 General check

If you have met a trouble with E391-library, first you must check the environmental variables. There are many variables that should be set correctly in order to make this system work. For example :

```
% printenv|grep E391;printenv|grep LD_LIBRARY_PATH=;printenv|grep CERN;pr
intenv|grep CLHEP
E391_HOME=/e391
E391_LEVEL=pro
E391_TOP_DIR=/e391/e391/pro
E391_LIB_DIR=/e391/e391/pro/lib
E391_CONFIG_DIR=/e391/e391/pro/src/config
LD_LIBRARY_PATH=.:/e391/e391/pro/lib/so:/e391/e391/pro/lib:/home/yamaga/lo
cal/lib:/usr/local/lib:/usr/lib:/lib:/cern/pro/lib:/cern/root/lib:/usr/ker
beros/lib
E391_DB_DIR=/e391/database
CERN=/cern
CERN_LEVEL=pro
CERN_ROOT=/cern/pro
CLHEP_BASE_DIR=/e391/local
CLHEP_LIB=CLHEP-g++.1.7.0.0

% printenv|grep MY
MY_TOP_DIR=/work/yamaga/dev/e391
MY_LIB_DIR=/work/yamaga/dev/e391/lib
MY_CONFIG_DIR=/work/yamaga/dev/e391/src/config
```

You should also make sure which command you are going to use, e.g:

```
% which eventDisplay
/work/yamaga/dev/e391/bin/eventDisplay
```

The command `ldd` is sometimes useful to check if correct shared-library is used.

```
% ldd /work/yamaga/dev/e391/bin/eventDisplay
        libHigzManager.so => /e391/e391/pro/lib/so/libHigzManager.so (0x40016
000)
        libE391Detector.so => /work/yamaga/dev/e391/lib/so/libE391Detector.so
 (0x403fe000)
        libunpack.so => /e391/e391/pro/lib/so/libunpack.so (0x40416000)
        libdetparam.so => /e391/e391/pro/lib/so/libdetparam.so (0x40439000)
        libdetector.so => /work/yamaga/dev/e391/lib/so/libdetector.so (0x4044
9000)
        libMidasManager.so => /e391/e391/pro/lib/so/libMidasManager.so (0x404
59000)
        libmidas.so => /e391/e391/pro/lib/libmidas.so (0x40468000)
```

```
        libcluster.so => /work/yamaga/dev/e391/lib/so/libcluster.so (0x404c20
00)
        libgufio.so => /e391/e391/pro/lib/so/libgufio.so (0x404e2000)
        libnsl.so.1 => /lib/libnsl.so.1 (0x404ec000)
        libpthread.so.0 => /lib/libpthread.so.0 (0x40503000)
        libkrb5.so.3 => /usr/kerberos/lib/libkrb5.so.3 (0x40519000)
        libk5crypto.so.3 => /usr/kerberos/lib/libk5crypto.so.3 (0x40571000)
        libcom_err.so.3 => /usr/kerberos/lib/libcom_err.so.3 (0x40583000)
        libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x40590000)
        libcrypt.so.1 => /lib/libcrypt.so.1 (0x4064f000)
        libm.so.6 => /lib/libm.so.6 (0x4067d000)
        libstdc++-libc6.2-2.so.3 => /usr/lib/libstdc++-libc6.2-2.so.3 (0x4069
f000)
        libc.so.6 => /lib/libc.so.6 (0x406e5000)
        libutil.so.1 => /lib/libutil.so.1 (0x40812000)
        libz.so.1 => /usr/lib/libz.so.1 (0x40815000)
        libdl.so.2 => /lib/libdl.so.2 (0x40823000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000
```

6.2 On development

If you met a trouble when you work on your local directory by using `get_packages` command, try to use another local directory as `MY_TOP_DIR`, and start from the beginning, i.e., from the `get_packages` command. The trouble might occur if the official-library is replaced with the latest version but your local directory still have the copy of the old version. Try to get as small number of packages as you can for your development.

6.3 Warnings in building on AIX

When you compile your program on the ps.cc.kek.jp, you will see many warning messages like

```
xlC_r4: 1501-245 Warning: Hard ulimit has been reduced to less than
RLIM_INFINITY.  There may not be enough space to complete the compil
ation.
xlf_r: 1501-245 Warning: Hard ulimit has been reduced to less than R
LIM_INFINITY.  There may not be enough space to complete the compila
tion.
```

Also when you link your executables with libraries on the ps.cc.kek.jp, you will see many warning messages like

```
ld: 0711-224 WARNING: Duplicate symbol: .std::basic_string<char,std
::char_traits<char>,std::allocator<char> >::basic_string()
ld: 0711-224 WARNING: Duplicate symbol: .std::basic_ostream<char,st
d::char_traits<char> >& std::operator<<<std::char_traits<char> >(st
d::basic_ostream<char,std::char_traits<char> >&,const char*)
ld: 0711-345 Use the -bloadmap or -bnoquiet option to obtain more i
nformation.
```

You can safely ignore these warning message. The former one appears due to the limitation to the resource usage for the user to avoid the serious trouble of operating system. It is written in the "CC User's Guide" on the web page of the KEK central computing facility. The latter one is a famous bug of the loader of the AIX, and it is known to be harmless in building the executables.

6.4 Contact

Questions, suggestions, requests or comments should be sent to
`yamaga@post.kek.jp`
`kensh@post.kek.jp`