CERN Computer Program Library Documentation

LHC++

C++ Libraries for HEP

Version 2.0.1

Applications for Physics & Infrastructure

Information Technologies Division

CERN, CH-1211 Geneva 23, Switzerland

Acknowledgements

HTL has benefited from the suggestions, advice and help of many individuals. Here is a list.
A special mention should be given to another list.

If you need help

The LHC++ Project page is available at the URL:

`http://wwwinfo.cern.ch/asd/lhc++/index.html`

The source of this document is marked up in XML using the DocBook DTD. It is translated with an XSLT style sheet into HTML and typeset using `xmltex` with LaTeX and the `docbook` class file developed at CERN to generate a printable PostScript file.

Contact Person: Andreas Pfeiffer /IT (`Andreas.Pfeiffer@cern.ch`)
Documentation: Michel Goossens /IT (`michel.goossens@cern.ch`)

Edition – June 2000 Typeset June 30, 2000

Table of Contents

List of Figures

Chapter 1: Presenting LHC++

1.1 A short historical introduction

In 1995 a working group, called *LHC++*, consisting of representatives of IT Division's ASD Group and of the various LHC experiments, was created to investigate how the approximate equivalent of the current CERNLIB environment could be provided in the LHC era. Although the primary objectives of the discussions concentrated on proposing solutions for future LHC experiments, it is essential that other HEP experiments which are coming online in the next couple of years will also be able to benefit from the proposed strategy.

It is clear that (industry) standard solutions should be used whenever possible. This is nothing new since in the past commercial libraries or programs, such as GKS/GTS, Numerical Algorithms Group (NAG) libraries, Phigs, Historian, and CMZ) have been used in the CERNLIB environment. Intensive investment of manpower should only be made in HEP-specific developments, where the needed functionality cannot be obtained *off the shelf* at affordable prices. An instance of such a HEP development is the CLHEP class library, which is also a nice example of reuse.

Working closely with the LHC collaborations and other HEP Laboratories in 1996 and 1997 an interim strategy was defined and a number of licences for the commercial components was obtained. When several *a priori* solutions were available, a study was made to determine which was the most suited in the HEP context. We also tested the proposed solutions at a few sites and ensured that it is a workable environment.

1.2 Objectives

The main objective of LHC++ is to satisfy the requirements of the LHC experiments in terms of the overall software environment that today is provided by CERNLIB, as requested in these experiments' *Computing Technical Proposals*

This translates into the following key points for the short to medium term:

- identify and provide key HEP-specific functions;
- define affordable solutions for the non-HEP-specific parts;
- monitor the non-HEP world for possible future useful software;
- study and understand the requirements of the LHC experiments for C++-based mathematical libraries, and evaluate existing and future developments, both commercial and otherwise in this area;
- study the requirements for a minimisation package (Minuit-replacement);
- undertake a number of pilot projects with the experiments to test the overall functionality of the LHC++ environment;
- supplement, where necessary, the existing documentation, both printed and online, with a set of user guides and tutorials;
- agree with the experiments and other HEP laboratories on a scheme for managing licenses, so that the best possible deals can be negotiated.

The scope of the LHC++ project covers the following:

- foundation level class libraries;

- mathematical libraries;
- graphical libraries;
- visualisation tool-kits, data analysis, histograms;
- event generators (in collaboration with, e.g., Lund);
- detector simulation (GEANT-4);
- object persistency (Objectivity/DB via IT/DB group at present).

The primary focus is on C++-based solutions, although, of course, developments in the software arena, in particular the increased importance of Java, are closely watched.

1.3 Collaboration with the experiments

The LHC++ project is a joint effort between IT Division (mainly ASD Group), the LHC experiments, plus NA45, Compass,... Andreas Pfeiffer is responsible for its overall coordination.

Regular (bi-weekly) LHC++ and RD45 meetings are held in Building 40. This is an ideal forum for communication with the physics community since its allows us to bring our users regularly up to date with the latest news. Also, and more importantly, it provides us with an input channel from the experiments about how they use the software, which are possible problem areas which have to be addressed, and what are the future developments they would like to see. The minutes as well as material presented on the progress in the various areas are available on the Web[1].

Twice a year a formal LHC++ Workshop[2] takes place where progress reports are presented by all HEP-wide collaborations who are using LHC++ software. This Workshop provides an efficient forum for feedback from the experiments and permits us to steer long-term development in the right direction by taking into account constraints and requirements of as wide a user base as possible.

1.4 Current Situation

By using commodity solutions where-ever possible, we ensure that the proposed solutions are widely used, well-debugged and well documented, and are also more affordable. These solutions are being complemented by HEP-specific components, where needed, by building a HEP user-layer on top of standards-conforming products.

To ensure that the commercial components work well together, the LHC++ strategy closely adheres to standards - both *de-facto* and *de-jure*. Examples of *de-jure* standards include the Standard C++ Library[3] and ODMG-compliant Object Database Management Systems (ODBMS), while instances of *de-facto* standards are industry standard graphics packages, such as OpenGL, Open Inventor, all originally from Silicon Graphics (SGI).

Table 1.1 below shows this layered structure more schematically. Licenses for all commercial components are available at CERN, where the software has been installed on mainstream Unix platforms (Linux, HP, and Sun) on AFS in `/afs/cern.ch/sw/lhcxx`, while on Nice/Windows/NT it is available under `z:\p32\lhcxx`.

All of the commercial LHC++ components come with excellent online documentation. In most cases, printed documentation, often in the form of published books, is also available and can be bought from the *Computing Bookshop* in IT Division at CERN. HEP-specific examples and other information specific to the HEP environment is clearly not available from the vendors.

[1] See URL `http://wwwinfo.cern.ch/asd/lhc++/meetings/index.html`.
[2] See URL `http://wwwinfo.cern.ch/asd/lhc++/workshops.html`.
[3] See URL `http://www.objectspace.com/Products/CCS/Standards/standards.html`.

Table 1.1: Layered structure of LHC++ system

GEANT-4, MCLIBS++
OpenGL, Open Inventor
HepFitting and GEMINI
NAG C library (with C++ headers)
CLHEP
HTL
HEPODBMS
ODMG, ODBMS (Objectivity/DB) + persistent STL
Standard C++ Libraries

The main purpose of the present manual is to provide a tutorial introduction to the use of LHC++ tools for physicist new to the LHC++ computing paradigm. The current guide describes the present state of some of the HEP extensions. Their precise form and application program interface (API) probably needs to be refined or extended in a few places. Therefore we invite all users of the LHC++ software to forward their comments and suggestions to the LHC++ team, preferably at the LHC++ regular meetings mentioned above.

1.5 An overview of the commercial components

The commercial components of LHC++ are chosen because they offer a coherent set of inter-operable solutions. They are built on standards and often come as part of the standard hardware or software bundled with the computer. Cost effectiveness has also been optimised both for CERN and for the general CERN HEP program participants.

1.5.1 OpenGL

OpenGL[4] is an industry standard for graphics. It is vendor-neutral and multi-platform, and is optimised for building environments for developing 2D and 3D visual applications. Several vendors already offer a hardware implementation of the standard, thus ensuring that rendering speed will be optimal.

The about 250 OpenGL procedures provide a wide range of graphics features, such as a set of geometric and raster primitives, various colour modes, display list or immediate mode, viewing and modelling transformations, lighting and shading, hidden surface removal and translucency, anti-aliasing, texture mapping, effects using fog, smoke, or haze, etc. As all licensed OpenGL implementations are required to pass a set of conformance tests, and implement the same specification and language binding document full portability between multiple platforms is guaranteed.

Documentation is available as two books: the *OpenGL Programming Guide*, and the *OpenGL Reference Manual*, both published by Addison and Wesley (available from the *Computing Bookshop* in IT Division).

1.5.2 Open Inventor

Open Inventor[5] is an object-oriented 3D toolkit to provide a comprehensive solution to interactive graphics programming. Its programming model is based on a 3D scene database optimised to ease building

[4] See URL `http://www.sgi.com/Products/Dev_environ_ds.html`.
[5] See URL `http://www.sgi.com/Technology/Inventor/index.html`.

graphics applications. It includes a large set of objects, such as cubes, polygons, text, materials, cameras, lights, track-balls, handle boxes, 3D viewers, and editors.

Open Inventor is built on top of OpenGL. It defines a standard file format (IV) for 3D data interchange and introduces a simple event model for 3D interaction. Animation is provided with *Engines*. Open Inventor offers a convenient multi-platform 3D graphics development environment, which allows efficient manipulation of objects in a windows and operating system independent way.

Open Inventor's IV files serve as the basis for the *VRML (Virtual Reality Modelling Language)* standard[6]. The Open Inventor toolkit is conveniently documented in three books *The Inventor Mentor*, *The Inventor Toolmaker*, and *The Open Inventor C++ Reference Manual* published by Addison-Wesley (available from the CERN Computing Bookshop).

1.5.3 Objectivity/DB, the Object Database

In order to study solutions for storing and handling the multi-Pbyte data samples expected with LHC, the *RD45 Project*[7] was established in 1995. The proposed solution should also be able to cope with other persistent objects, such as histograms, calibration monitoring data, etc. It was found that the best candidate for handling this problem is a *ODMG* (Object Database Management Group)[8], compliant object database used together with a mass storage system, based upon the IEEE reference model for mass storage systems. After considering a few alternatives, the presently favoured solution uses *Objectivity/DB*[9] and *HPSS* (High Performance Storage System)[10].

The Objectivity/DB database provides object-persistency services for GEANT-4 and the experimental data. It must fully support HEP meta-data, not only the persistent data collections themselves, but it must also handle the selections producing these collections, and the predicates themselves. Replication of large database images on local area and wide area network configurations containing heterogeneous hardware must allow collaborators all over the world to actively participate in data analysis. Given the large time scales (the lifetime of the LHC software will span at least twenty years) schema evolution and versioning are important aspects which must be taken into account. Objectivity/DB comes with a set of administrative tools to ease database management. Objectivity/DB also comes with an *Advanced Multi-threaded Server* (AMS) and an interface to HPSS. This provides fast access to data and offers a large performance improvement when updating data stored in remote databases. It is expected that each experiment will run starting in 1998 a production service of one or more of these servers.

Objectivity/DB has a layered logical storage level, with at its top the *federated database*. Each federated database logically contains one or more *databases*, with the latter containing the *objects*, which are clustered, for efficiency, inside *containers*. An *object* itself consists of standard C++ constructs, variable-size arrays, relationships and references to other objects, and type constraints. Persistent objects can be created and deleted dynamically by any application. The data model, or *schema* is stored inside the federated database.

1.5.4 Mathematical Libraries

CERN no longer has any in-house mathematician supporting mathematics libraries. Therefore, we shall have to rely on libraries developed outside CERN, and it was decided to make the *NAG C-language*[11] library available. Although the NAG C Library provides the basic functionality required by HEP, a small number of routines (basically special functions) are currently unavailable. A future release of the above library is likely to incorporate these routines.

---

[6] See URL `http://vrml.wired.com`.
[7] See URL `http://wwwinfo.cern.ch/asd/cernlib/rd45/index.html`.
[8] See URL `http://www.odmg.org/`.
[9] See URL `http://www.objy.com/`.
[10] See URL `http://www.sdsc.edu/hpss/`.
[11] See URL `http://www.nag.co.uk/numeric/CL.html`.

1.5.5 Statistical Data Analysis: the Gemini package

Gemini is a GEneral MINImization and error analysis package implemented as a C++ class library. Minuit's functionality is provided in a *Minuit-style* (even if, internally, another minimizer may actually do the work) and new functionality offered by NAG C minimizers is added. Gemini thus provides a unified C++ API both to standard Minuit and to NAG C family of minimizers. For the common subset of functionality, it is up to the user which minimization engine does the work: Minuit or NAG C. The user can easily switch between various minimizers without essential changes in the application code. The currently supported set of minimizers (Minuit and NAG C) can be extended without essential changes in the API.

The abstract base class `GEmini` defines an interface to the common functionality. The `CMinuit` class is derived from `GEmini` and provides a Minuit-based implementation of the GEmini functionality plus Minuit-specific extensions. Similarly, the `NAGmin` class is derived from `GEmini` as well and provides a NAG-based implementation of the `GEmini` functionality plus NAG-specific extensions.

There is no single class which contains references both to Minuit and to NAG C, so that orthodox Minuit or Nag C users are not forced to link the other library.

Gemini finds a minimum of an objective function, possibly subject to general constraints, and performs an error analysis. The concept of errors is that of Minuit, so that it is the user's responsibility to properly scale the inversed Hessian, and to properly interpret the results. Both Hessian based errors and Minos errors are implemented. Correspondingly, two types of function contours (or confidence regions, in statistical problems) are available: elliptical and Minos ones. Minos error analysis is, however, possible only for bound constraint problems.

1.6 Overview of the HEP-specific components

Although commercial and public-domain packages offer a great deal of functionality, there is a clear need to supplement them with HEP-specific extensions. Some of these extensions take the form of complete class-libraries, such as CLHEP. Others represent large toolkits, such as *GEANT-4*[12], in areas such as graphics and visualisation, the basic tools, such as Open Inventor for basic 3D graphics and Qt for 2D graphics for visualisation, need to be extended to cope with the specific needs of the HEP experiments.

1.6.1 HEPODBMS

HepODBMS is a set of class libraries built on top of the ODMG C++ interface. Their purpose is to provide a higher level interface than is specified by the ODMG, to simplify the porting of existing applications and provide a minimum level of support for transient-persistent switching. Furthermore, these libraries help to insulate applications against changes between releases from a given vendor and between the products of different vendors.

1.6.2 CLHEP

The *CLHEP* project[13] was initiated at CHEP'92; it intends to provide *foundation level* classes required in HEP. At present they include:

- `Alist` for lists and list iterators;
- `Combination`;
- `Geometry` for vectors, rotations, transformations;

[12]See URL `http://wwwinfo.cern.ch/asd/geant4/geant4.html`.
[13]See URL `http://wwwinfo.cern.ch/asd/lhc++/clhep/index.html`.

- `Matrix` for matrix manipulations;
- `Random` for random numbers;
- `String` for different string types;
- `Units` for system of units and physical constants;
- `Vector` for vector operations (3-vector and Lorentz-type).

CLHEP became formally part of LHC++ in 1995. The first official release of CLHEP (V1.0) took place in April 1997 (CHEP'97). CLHEP-based classes will be integrated in the beta-release of GEANT 4 early 1998. The complete user documentation, with a detailed description of all classes is being written and will be available by the end of 1997.

1.6.3 The HTL class library

The HTL class library provides Object Oriented histograms. They come in two versions:

- Persistent histograms (based on Objectivity/DB)
- Transient histograms (text file I/O)

XML based persistency for Transient HTL is currently under study.

1.6.4 HepVis

The goal of the *HepVis Project*[14] is to create and distribute a toolkit library consisting of graphical objects capable of representing the most common entities of a collider physics experiment. Previous experience has shown that mere representations of objects on a workstation screen is insufficient, and that native support for picking objects with user-defined actions, and a high-degree of interactivity, both local and global, is needed. Therefore, the HepVis toolkit is being implemented as an extension to Open Inventor, providing common physics objects as subclasses or as real extensions. Only the graphical representation of the objects will be defined, leaving it up to the experiments to define physics objects and their behaviour, and whether to integrate these with the graphical objects in question.

1.7 Problem Reporting

Due to the phase out of the GNATS problem reporting system at Cern, users should submit their problem reports by sending a mail to `Heplib.Support@cern.ch`.

[14] See URL `http://physics.web.cern.ch/Physics/Workshops/hepvis/`.

Chapter 2: LHC++ Object Model

2.1 A historical digression: Ntuples and PAW

During the past decade, many HEP experiments have based their interactive data analysis on the following steps (see the top half of Figure 2.1).

1. *Raw and reconstructed* data are stored in *banks* in an experiment-specific hierarchical format. Most of the time one uses many different files on several distinct hosts.

2. These data are *distilled* and *reclustered* to obtain a more compact and thus more efficient representation. This permits a significant speed-up for the down-stream analysis compared to using the data in their raw form as described in point 1. This format corresponds to the so-called HBOOK Ntuples. One drawback of this method is that the direct relation to the raw *event data* is lost.

3. Ntuple files are analysed *interactively* with programs like PAW. Plots of physics variables are produced by extracting information contained in one or more of the Ntuple rows or columns, binning them in HBOOK histograms and then operating on these histograms to obtain the best representation.

The advantage of Ntuples is that their format is known and simple enough, so that a general purpose analysis tool, such as PAW can cope with data coming from any experiment. On the other hand, since no link to the original data exists, Ntuples impose a limitation on the structure of the data physicists can use for their analysis. On top of that, since the data were copied from the original files into a dedicated Ntuple file, each time original dataset changed most Ntuple files had to be regenerated.

Two kinds of Ntuples exists. *Row-Wise Ntuples* transform a complex data structure into a simple tabular form. *Column-Wise Ntuples* on the other hand improve the flexibility of the Ntuple data model by allowing the definition of variable-length items, but they still are difficult to use to describe complex data structures, like those of the reconstructed data. Moreover, the Ntuple Query language is rather non-intuitive and complex to master.

2.2 The new Data Model

Most new HEP experiments assume that it will be possible to make both raw data and reconstructed data available *on-line* thanks to the integration between Objectivity/DB and HPSS. Each experiment will have its own data model and physicists should be able to *navigate* through it. This is a major problem for a general-purpose Interactive Analysis environment, since, unlike the Ntuple case, there no longer exists

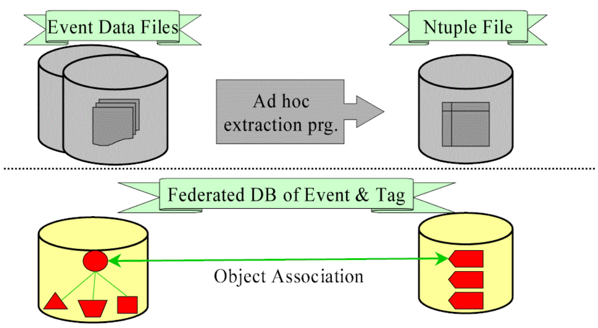

Figure 2.1: The Ntuple and TagDB models

a common and pre-defined data model shared amongst all experiments. This problem can be solved but there is no easy way out: the general-purpose tool should be able to access the arbitrary experiment data model using some kind of run time type information or the initial data model definition.

Since all data is supposed to be *on-line*, the role of the Ntuple replacement could be quite different. While reasonably small *personal* data collections will still exist, the main concern will probably be how to index large event stores to speed up the analysis.

The RD45 Project suggested one approach to deal with both problems. The idea it to speed up queries by defining for each event a *Tag*, i.e., a small collection of its most important physics attributes plus an association with the event where the Tag data come from. Such *Concrete* Tags contain copies of data members of a persistent data class. A collection of tag objects is saved together in a Tag Database, something intermediate between an Event Directory and an Ntuple. Since they are globally defined for the whole experiment, concrete tags can be optimized so that they offer a very efficient way to make initial cuts on attributes, thus achieving a high degree of selectivity. On top of that, at any moment you are able to cross the association to the event if you want to retrieve any other details about the full event, which are not contained in the Tag (see bottom part of Figure 2.1).

The *top part* of the figure shows schematically the present *traditional* approach. The event data (raw, reconstructed events, calibration constants, etc.) are represented at the left hand side . They are distributed over many files residing on various hosts. An *ad-hoc* program reads a set of interesting quantities in these files and writes the retrieved information into an Ntuple file. In this Ntuple file data are reclustered most of the time according to a simple table structure (more complex arrangements are, of course, possible). As the Ntuple file format is known, interactive visualisation programs can efficiently read these data files, thus allowing a fast and convenient physics data analysis system to be set up. Note, however, that these Ntuple files are completely disconnected from the original data, so that it is impossible to automatically update in the Ntuple files information which changes in the original. Also, it is not possible to retrieve transparently from the original files data which were not saved in the Ntuple when it was created.

The *lower part* of the picture figure what the situation looks like in the Tag model. In this case often-used data are once more reclustered (using experiment-wide concrete tags or user-defined generic tags), but all data remain inside the same federated database, and there exists a bidirectional link between the reclustered and the original data. In this way, when the original data are reprocessed, it is trivial to update the tag data, so that they remain always up-to-date. Conversely, when for a given event the information in the tag database is not sufficient, then the link to the complete event data allows you to retrieve the supplementary information in a convenient and straighforward way *on the fly*.

In general the selection of *key* attributes characterising events will be made by the experiment or group, so that concrete tags are mostly defined for experiment-wide or workgroup-wide data sets. However, individual physicists have the possibility to define their own simpler data collection by using the *Generic Tag* mechanism. This second light-weight procedure allows you to define a tag *on the fly*, without creating a persistent class. Compared to the concrete tag, there is, of course, a small performance penalty, but this is most of the time balanced by an increased flexibility, since at any time new fields can be added to the tag and the association to the complete event data remains available. Presently, both the concrete tag and the generic tag are defined in a C++ program before being used in the Interactive Analysis framework.

2.3 Implementing the Data Model: explorable collections

When creating a tag (either generic or concrete) a description of its fields (name and type) must be provided. This information is used later to access the data. The set of individual tags is called an *Explorable Collection*, i.e., a collection of objects implementing an access interface suitable for an Interactive Analysis Tool.

Chapter 3: Setting up the user environment

The LHC++ environment is defined by a set of environment variables which define where to retrieve tools, libraries , include files etc. Depending on the flavour of the Unix shell used (Bourne o csh), different commands are used to define such variables. In order to have the environment properly set up at any new login, those commands must be recorded in a startup script which is then executed when the user logs in. The name of these script depends on the shell flavour as well.

3.1 Environment variables required by LHC++

These are the environment variables LHC++ relies on:

- `LHCXXTOP`, the location where LHC++ is installed;
- `PLATF`, the specific computing platform ;
- `LHCXX_REL_DIR`, the release of LHC++ in use;
- `OBJY_VERS`, the release of Objectivity/DB in use.

These environment variables are used to locate all LHC++ components. LHC++ provides startup scripts to set reasonable defaults for those variables, but in some cases the user may want to override them. The next sections will explain how to setup the user environment depending on the flavour of the active shell. *Shell Support - tcsh and zsh for pedestrians*[1] provides a comprehensive description of two mainstream shells (tcsh and zsh).

3.2 Setting up the environment for Bourne shell flavour shells

The Bourne family of Unix shells includes sh, bash, ksh and zsh. The LHC++ environment includes a startup script users have to source to get the environment set up. This script must be sourced from the `.profile` system script. The startup script will define all the above mentioned environment variables unless they're already defined. The detection of the plaform in use relies on the well known environment variable `OS`. If that variable is not defined and the user did not define `PLATF` beforehand, the script will assume that the platform is Linux Red Hat 6.1. To source the LHC++ script, add these lines to `.profile`:

```
# At Cern LHCXXTOP=/afs/cern.ch/sw/lhcxx
export LHCXXTOP=<where LHC++ is installed>
# Source the startup script. If using versions other than 2.0.1
# just substitute it with the new version.
. $LHCXXTOP/share/LHCXX/2.0.1/install/sharedstart.sh
```

Please do not define `PLATF` as the AFS variable `@sys`. since such variable can be ambigous.

3.3 Setting up the environment for csh shell flavour shells

The csh family of Unix shells includes csh and tcsh. The LHC++ environment includes a startup script users have to source to get the environment set up. This script must be sourced from the `.login` The startup script will define all the above mentioned environment variables unless they're already defined. The detection of the plaform in use relies on the well known environment variable `OS`. If such variable is not defined and the user did not define `PLATF` beforehand, the script will assume that the platform is Linux Red Hat 6.1. To source the LHC++ script, add these lines to `.login`:

[1] See URL `http://consult.cern.ch/writeup/shellsintro/`.

```
# At Cern LHCXXTOP=/afs/cern.ch/sw/lhcxx
setenv LHCXXTOP <where LHC++ is installed>
# Source the startup script. If using versions other than 2.0.1
# just substitute it with the new version.
source $LHCXXTOP/share/LHCXX/2.0.1/install/sharedstart.csh
```

Please do not define `PLATF` as the AFS variable `@sys`. since such variable can be ambigous.

3.4 How to get a fresh Objectivity/DB federated database

To use some of the LHC++ components, Objectivity/DB is needed. If you do not yet have such a database set up, you should contact your experiment's or group's Objectivity/DB coordinator, who will take the necessary steps to register you for database use. You will then be assigned a *Federated Database IDentifier* (FDID), to uniquely identify your federated database to the Objectivity/DB servers. This number should be specified when first installing the database, as described below.

When you log on you must check whether the lock server is running. You can do that with the command: `oolockmon`. If it is NOT running, you may start it with the command: `oolockserver -noauto`. If it is running and you want to kill it you issue the command `ookillls` .

To install your own user federated database you can do the following (Bourne flavour shells):

```
export MY_FDID=<a valid number in the range allocated to you>
export OO_FD_BOOT=<name of your choice; preferably full path>
$HEP_ODBMS_DIR/etc/getdb $LHCXXTOP/share/HTL/1.1.1.1/schema/HISTO $OO_FD_BOOT $MY_FDID
```

For csh flavour shells:

```
setenv MY_FDID <a valid number in the range allocated to you>
setenv OO_FD_BOOT <name of your choice; preferably full path>
$HEP_ODBMS_DIR/etc/getdb $LHCXXTOP/share/HTL/1.1.1.1/schema/HISTO $OO_FD_BOOT $MY_FDID
```

Once you got your database, do not forget to add the command defining where federated database is to the proper system setup script, i.e. :

For Bourne flavour shells add to `.profile`

```
export OO_FD_BOOT=<name of your choice; preferably full path>
```

For csh flavour shells add to `.login`

```
setenv OO_FD_BOOT <name of your choice; preferably full path>
```

To check wheter the database has been installed properly, try:

```
oodumpcatalog $OO_FD_BOOT
```

3.5 Installation troubleshooting

Installation procedure failure is usually due to one of the following reasons.

- *Missing access rights to the Objectivity package.* In this case you should get your Objectivity licence before trying again.
- *Lockserver running in autorecovery mode.* Use the Unix ps command to find out the lockserver's running mode:

```
ps -ef | grep ools
```

  If you see something like:

```
dinofm 20536 1 0 Oct 27 ? 1:37 ools
```

  restart the lockserver in the proper mode:

```
> ookillls
> oolockserver -noautorecovery
> ps -ef | grep ools
```

  where the ps command should show something like:

```
dinofm 20536 1 0 Oct 27 ? 1:37 ools -OO_NO_AUTOREC
```

Chapter 4: Accessing the Objectivity/DB database

4.1 What does the user have to know about the database?

Ideally, the hierarchical structure of the data storage should be completely transparent to the average user. Therefore, the LHC++ Team has done its best to hide the impact of the database on the C++ user code to a minimum. However, it is best that the users of the modules are aware of some basic principles, and how they relate to the experimental data model used by the various analysis programs.

Let we have a look at Figure 4.1. It shows the storage hierarchy used to store event data at the left, together with the user's view of these data at the right.

We start with the user's view (right hand side of the picture). The user likes to think in terms of events (the octagons), and wants to deal with, for instance reconstructed tracks (the triangles), hits in the forward calorimeter (the diamonds), or the calibration for the TPC (the pentagons), etc. Users should not be directly concerned (apart perhaps for efficiency considerations) how these various data elements are actually stored in files and distributed over a network. They prefer to have a *logical* view of their event and navigate between its various componenents in a transparent way. It is up to the data administrator to make sure that the data are stored in a way optimising performance and throughput for the end user.

This is possible using an object oriented database system, such as Objectivity/DB (left hand side of the picture). All data are kept in one *federated database*, which is basically just a file containing the catalog of the database files and the hostnames where they reside. It also contains the *schema* (object model) used by the data in the various databases.

The *databases* themselves are also separate files, which can reside on different nodes and they can consist of multiple *containers*, that can be thought of a contiguous areas on a file.

Finally, each container consists of one or more *persistent* objects (e.g., histograms, reconstructed tracks, fits). As seen in the picture, the mapping of the event to its components is very flexible, allowing different parts of an event to reside in different containers, and/or databases (even on remote nodes). Moreover, since the end users only access the full data through the logical structure, they are never affected by changes in the physical layout of the database.

4.2 Getting access to an Objectivity/DB database

In 1999 a central service will be run for the various experiments using Objectivity/DB to provide access to their databases from various platforms using AMS. However, at present, it is necessary to associate a federated database with a given machine, so that you must always connect to that same machine if you want to access that database.

In particular, when working on a cluster (such as hpplus) you must remember the real node name (e.g., `hpplus16`, as in the example in the previous chapter) of the machine which you used to create the database. When you want to use the database in a later session, you must always connect to that node, otherwise you will not be able to access your data (unless your experiment is running Objectivity/DB's AMS, a facility to share databases across the network).

The first step (after creating the database) is to tell Objectivity/DB where it can be found. This is done with the environment variable `OO_FD_BOOT`, which should be set to the full path name of the boot file of the database that you want to access. This boot file is actually a small ASCII file, which contains valuable information about your federated database.

```
> more $OO_FD_BOOT
ooFDNumber=30500
ooLFDNumber=65535
ooPageSize=8192
```

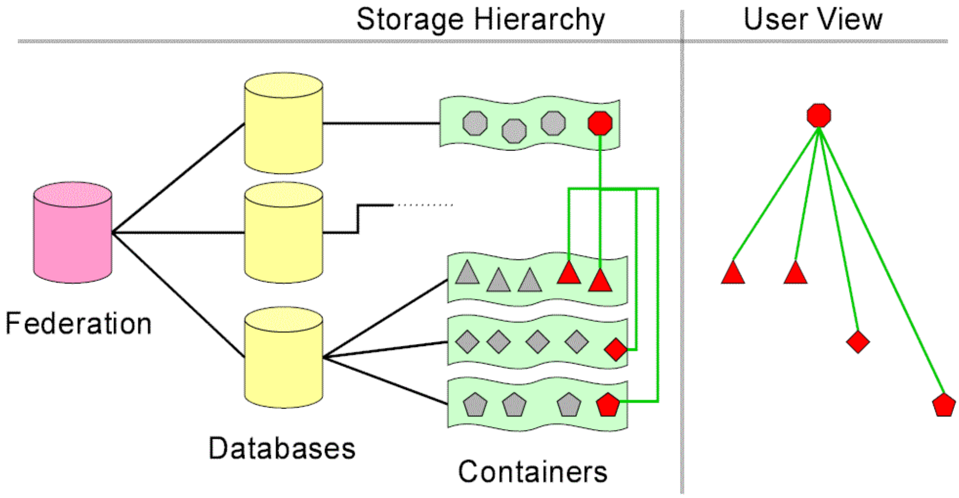

Figure 4.1: The storage hierarchy and the user view

```
ooLockServerName=hpplus16
ooFDDBHost=hpplus16
ooFDDBFileName=/afs/cern.ch/user/g/goossens/HP-UX/explorer/HEPEXP.FDDB
ooJNLHost=hpplus16
ooJNLPath=/afs/cern.ch/user/g/goossens/HP-UX/explorer
```

You should *NEVER* change its contents!

In particular, changing the path of the federated database in this file after you moved it will *not* work.

Once the `OO_FD_BOOT` variable is set correctly, you can run the Objectivity/DB management tools. For instance the program `oodumpcatalog` displays the catalog, showing the different databases associated to the current federated database.

```
> oodumpcatalog

Objectivity/DB (TM) List Database Files Utility, Version 4.0.2
Copyright (c) Objectivity, Inc 1990, 1996. All rights reserved.

FD Name   = HEPEXP
FD ID     = 30500
FD File   = hpplus16::/afs/cern.ch/user/g/goossens/HP-UX/explorer/HEPEXP.FDDB
Boot File = hpplus16::/afs/cern.ch/user/g/goossens/HP-UX/explorer/HEPEXP
Jnl Dir   = hpplus16::/afs/cern.ch/user/g/goossens/HP-UX/explorer
Lock Host = hpplus16

DB Name   = SimpleTestDatabase
DB ID     = 3
DB Image  =
hpplus16::/afs/cern.ch/user/g/goossens/HP-UX/explorer/SimpleTestDatabase.HEPEXP.DB
```

Thus, in the example above, we see that we need to connect to the node `hpplus16` (the so-called *Lock Host*). It should also be noted that each federated database should have a different federated database

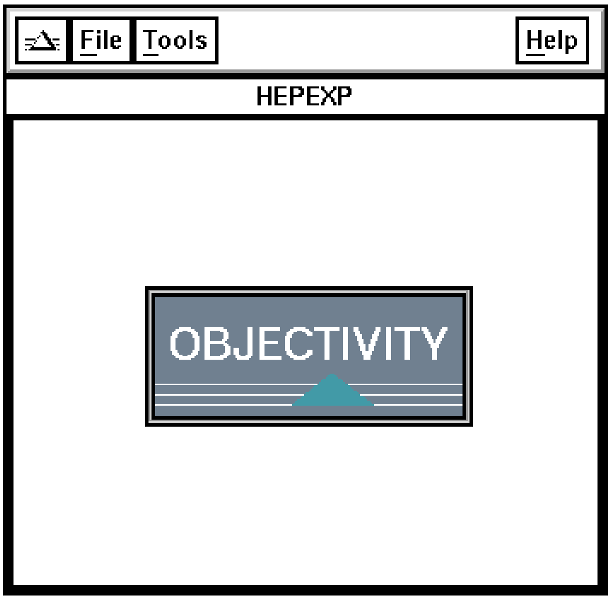

Figure 4.2: The `ootoolmgr` initial panel

number to enforce proper locking management when the same lockserver is used by more than one federation (e.g., when the lockserver is running on a central service, such as hpplus). A series of federated database numbers have been allocated to experiments and user groups (see *Draft recommendations concerning Objectivity/DB DBA issues*[1] for a proposed list). As explained above, when registering as a database user with your experiment's or group's Objectivity/DB's coordinator, you get a unique number assigned to ensure the uniqueness of the federated database number.

4.3 Accessing the Objectivity/DB from inside a C++ program

After the initial installation, you have a federated database, which has no associated databases yet. This can be seen by using the Objectivity/DB tool `ootoolmgr` (`oobrowse` on Windows/NT), which allows you to browse the contents of all databases (down to the object level) in a federated database. If you want to run `ootoolmgr` just type

```
> ootoolmgr
```

on the command line. You will then get an Objectivity/DB panel, as shown in Figure 4.2.

Then in the `File` pull-down menu you choose the database (most of the time it is enough to click the default setting at the bottom, which corresponds to the database selected with the `OO_FD_BOOT` environment variable). Then you can go to the `Tools` pull-down menu and choose `Browse FD` at the top. Then you will see the `Objectivity/DB - Browse FD` appear, with four main windows, namely the names of the *Databases*, *Containers*, *Basic Objects*, and finally, at the bottom, the contents of the selected object. Just after initialisation, there are no databases yet, as seen in Figure 4.3.

Before writing our first C++ program to use the database, let us first establish three basic rules.

1. One must first establish a connection to the database with the `Init` method.

[1] See URL `http://wwwinfo.cern.ch/asd/cernlib/rd45/recommendations/dba.html`.

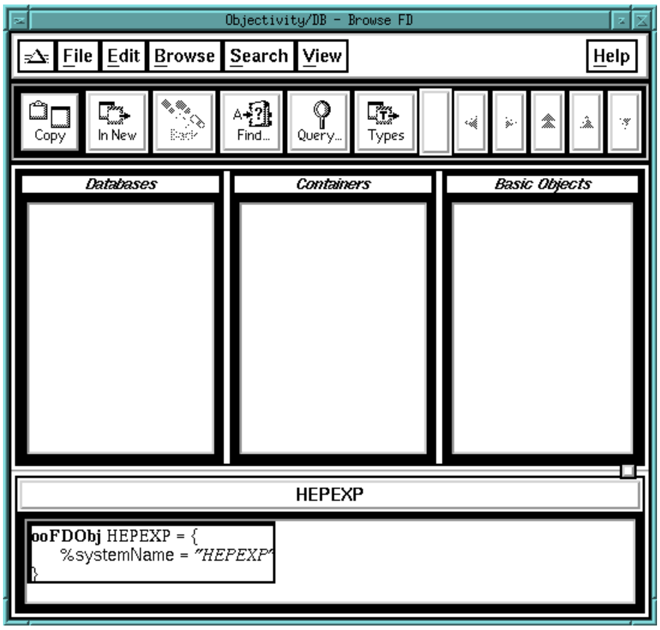

Figure 4.3: Using the tool `ootoolmgr`

2. *Transactions* are used to retrieve or store persistent objects.

   - A transaction is set up using the `startUpdate` or `startRead` methods.
   - A transaction is terminated using the `commit` or `abort` methods. The `commit` method will save all changed objects in the database, whereas `abort` will revert the database to the state before the transaction was initiated.

3. To access persistent objects *smart* pointers are provided. They are indistinguishable from normal C++ pointers, they are merely declared with a different syntax using `HepRef`, e.g., for a 1D histogram we would declare

   ```
   HepRefP(Histo1D) myHisto (...);
   ```

   while for a pointer to a non-persistent C++ object you would write

   ```
   Histo1D *myHisto ...
   ```

   Once you have declared your smart pointers to your persistent object, all navigation between objects is completely like in the case of normal pointers; whenever a reference is made to a smart pointer, a callback to the database will automatically fetch the required data. Moreover at the end of the transaction all modified objects will automatically be stored in the database at commit time.

If, while reading through the examples, you want to know more details about the HepODBMS classes, you should consult the *Reference Manual for the HepODBMS package*[2].

[2]See URL `http://wwwinfo.cern.ch/asd/lhc++/HepODBMS/reference-manual/HepODBMS.html`.

4.4 Manipulating the Objectivity/DB database and its containers in a C++ program

To show a few more of the class methods available to manipulate an Objectivity/DB database, we can look at the following code.

```
/* dbAccess.cpp */
#include "HepODBMS/tagdb/HepTagDbApplication.h"
#include "HepODBMS/tagdb/HepEvent.h"

class dbAccessApp : public HepTagDbApplication {
  // Application inherits session control from HepDbApplication
public:
  // this application implements just one method: run the
  dbAccessApp(const char *name) : HepTagDbApplication(name)
  {};

  int run()
  {
    // print an
    message("about to initialise the db connection");
    Init();        // initialise the db session
    message("starting an update transaction");
    startUpdate(); // start an update transaction

    // create a new database (file)
    HepDatabaseRef  myDb = db("MyDatabase");

    // if the database ref is not valid:
    // - print a message
    // - exit the application with an error code
    if (myDb == 0)
      fatal("could not find or create MyDatabase");

    // create a new container in this database
    HepContainerRef cont = container("MyContainer");
    if (cont == 0 )
      fatal("could not find or create MyDatabase");

    // work with the container and database
    // (e.g. create histograms, tags or other persistent objects)

    for (short i=0; i<1000; i++)
    {
      // create a new event in my container
      HepRef(HepEvent) event = new(cont) HepEvent;
      if (event == 0)
        fatal("could not create a new event");
    }
    message("created 1000 events");

    printContainerMap( ) ;

    // delete the container from the database
    // including all events
    HepDelete(cont);
    warning("deleted the container");

    // delete the database from the federation
    HepDelete(myDb);
    warning("deleted the database");
```

```
    // commit all changes made during this transaction
    commit();
    return 0;
  }

};

int main(int argc, const char *argv[])
{
  dbAccessApp myApp(argv[0]);  // create an application object
  return myApp.run();    // call it's run method
}
```

The `HepDbApplication` class defines the transaction methods `abort`, `commit`, `startRead`, `startUpdate` (described already in Section 4.3), as well as four methods for sending an informative string to the user console: `fatal` (prints a fatal error message and aborts), `error` and `warning`, (prints an error and warning message and continue), and `message` (just prints a message).

The `HepDatabaseRef` declaration sets up a database handle `myDb` using the `db` method from the `ooSession` class. Once we have opened a database, we declare a container with `HepContainerRef`, which returns us a handle `cont` using the `container` method from the `ooSession` class. The container handle is then used to store one thousand events of type `HepEvent` inside the `for` loop. Just before we delete our database we print a map of the containers with the `printContainerMap` method, which shows the containers together with their object identifiers (the `HepSystem` and `ExplorableDescr` containers are created when HEPODBMS is set up in the `System` database). Finally, we delete the container and database by specifying their respective handles to the `HepDelete` method. Below, the output generated by the above C++ code is shown.

```
dbAccess: about to initialise the db connection
dbAccess: starting an update transaction
dbAccess: created 1000 events
HepSystem --- #3-4-3-1
ExplorableDescr --- #3-5-3-1
MyContainer --- #12-3-1-1
WARNING: dbAccess: deleted the container
WARNING: dbAccess: deleted the database
```

4.5 Objectivity/DB administration tools

Objectivity/DB provides a whole set of administration tools to manage a federated database. These tools are described in detail in the *Objectivity/DB Administration* manual. In this section we briefly describe the more useful from the physicists' point of view.

- `oodumpcatalog` provides summary information about a federated database;
- `ootoolmgr` (`oobrowse` on Windows/NT) allows you to browse federated database schema and data;
- `oocleanup` resets pending locks;
- `oodeletedb` deletes a *physical* database.

Chapter 5: Histogram and tag classes

All classes are being documented and a user guide and reference manual for the available components will be available soon. In this chapter we shall describe the main characteristics of the HTL classes and also learn about the tag classes. As explained before, ready-to-run examples can be found in the following two directories.

```
# At Cern LHCXXTOP=/afs/cern.ch/sw/lhcxx
$LHCXXTOP/share/HepODBMS/<version>/HepODBMS/examples
$LHCXXTOP/share/HTL/<version>/HTL/examples/
```

Each example has its own subdirectory (corresponding to the first part of the filename, i.e., preceding the suffix `.cpp`, specified as a comment on the first line of the examples' C++ code in this manual), which contains the C++ source code in a file with extension `cpp`, as well as a `GNUmakefile` which will compile, link and generate an executable if run with `gmake`. These example programs should form an excellent basis to get started writing your own application programs.

5.1 The HTL package

The Persistent HTL Package provides the basic histogramming functionality of HBOOK along with some additional features. See *HTL - The Histogram Template Library*[1] for more details.

5.2 Converting an HBOOK file into an Objectivity/DB database

The program `Hbook2Objy` converts histograms contained in an HBOOK file into persistent HTL histograms that can be saved in an Objectivity/DB database. Only one- and two-dimensional histograms are converted, Ntuples are ignored. To execute the program you should type:

```
$HISTOODIR/bin/Hbook2Objy <hfile> <OBJ-federated-DB>
```

Starting from the histograms in the HBOOK file `<hfile>` the program will create a new database with as name the name of the HBOOK file `<hfile>` inside the Objectivity/DB database pointed at by the environment variable `OO_FD_BOOT`. If `<hfile>` already exists, then you will be asked whether you want to overwrite the original file.

It should be noted that the hierarchical (directory) structure of the HBOOK file is not preserved. Nevertheless, since the name of each created persistent histogram corresponds to the full path name inside the HBOOK hierarchy, it is easy to distinguish between histograms with the same name but coming from different HBOOK directories. An example of the translation of a set of HBOOK histograms is shown in Figure 5.1. At the top we see some (1D and 2D) histograms with the HBOOK Directory Browser, and at the bottom the same histograms after conversion as viewed from the Objectivity/DB database with the `HistOOgramReader` module.

5.3 The tag classes

5.3.1 Writing tags

As explained earlier (see Section 2.2), tags are a small collection of the most important physics attributes of an event plus an association with the event in question. Tags provide a natural and efficient syntax

[1] See URL `http://wwwinfo.cern.ch/asd/lhc++/HTL/index.html`.

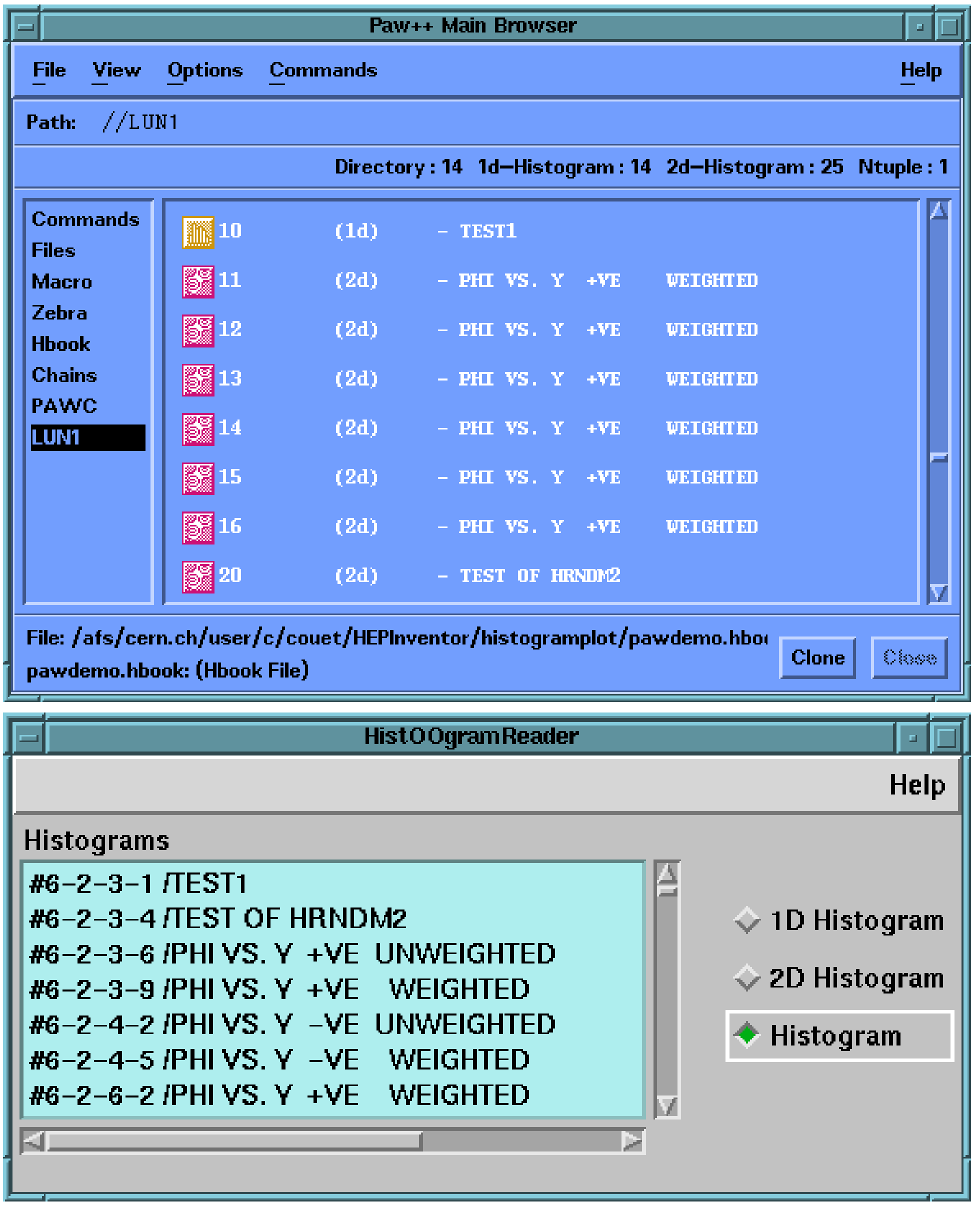

Figure 5.1: Transforming HBOOK histograms into HTL histograms

for handling event data, speeding up queries, cuts handling, etc., substantially, while at the same time offering the possibility to easily access the original full data. Usually, the event tags are chosen in such a way that a high degree of selectivity can be obtained by first cutting on attributes in the event tag, while, if needed, the association to the full event can be exploited in order to retrieve any other information not contained in the tag.

In workgroup-wide data sets, individual physicists should still be able to have their own simpler data collection, so an easy-to-use *Generic Tag* is defined as well. Presently, both the Event Tag and the Generic Tag have to be created inside a C++ program before using the Interactive Analysis framework.

The data types that can be stored in a tag are `long` and `short` integers, `float` and `double` real numbers, and an 8-bit `char`.

Examples of the use of tags are given with more details in the HEPODBMS documentation[2].

5.3.2 Converting HBOOK Ntuples to Objectivity/DB

The LHC++ environment provides tools to convert existing HBOOK Ntuples to Objectivity/DB[3].

[2] See URL `http://wwwinfo.cern.ch/db/objectivity/docs/hepodbms`.

[3] See URL `http://wwwinfo.cern.ch/asd/lhc++/TOOLS/ntkit/ntupleconvnew.html`.

Chapter 6: Glossary

AMS Advanced Multi-threaded Server (Objectivity/DB).

AFS Andrew (distributed) Filesystem.

CORBA Common Object Request Broker Architecture, from the OMG.

HPSS High Performance Storage System. A high-end mass storage system developed by a consortium consisting of end-user sites and commercial companies.

NFS Network Filesystem, developed by Sun.

Objectivity/DB Vendor of an ODBMS. Chosen at CERN in the framework of RD45.

ODBMS Object Database Management System.

ODL Object Definition Language. Specification language defining the interface to object types conforming to the ODMG Object Model.

ODMG Object Database Management Group. Develops standards for ODBMSes, e.g., in the area of scalability, heterogeneity, WAN support (distribution, replication, caching, recovery, etc.), and schema evolution.

OMG Object Management Group. Consortium of over 400 members from the software, hardware, and large end-user communities, whose goal is to standardise and promote object technology in all forms, in particular by proposing specific standards which should increase portability of customer software across ODBMS products.

ORB Object Request Broker

GEANT-4 Object-Oriented Toolkit for Simulation in HEP.

RD45 Project Research and Development project to investigate object persistency for HEP.