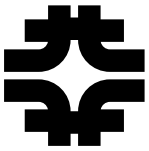

Fermi National Accelerator Laboratory

FERMILAB-Conf-98/322

The ZOOM Fermilab Physics Class Libraries

Mark Fischler, Walter Brown, Philippe Canal and John Marraffino

Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

November 1998

Presented at the *International Conference on Computing in High Energy Physics (CHEP '98)*, Chicago, Illinois, August 31-September 4, 1998

Operated by Universities Research Association Inc. under Contract No. DE-AC02-76CH03000 with the United States Department of Energy

The ZOOM Fermilab Physics Class Libraries

Mark Fischler, Walter Brown, Phillippe Canal *and* John Marraffino
Fermilab

October 12, 1998

1 Purpose and Structure

Several years ago, the two major collider experiments at Fermilab (DØ and CDF) decided that new software development for Run II will be largely done in C++. The run is slated to start in 1.5 years, an aggressive time frame for a major change in development language and style. If despite the transition each experiment (and sometimes multiple groups within an experiment) were to develop each needed module, the C++ strategy would not be advantageous. Thus it was deemed useful to have a library development group specifically responsive to Run II needs. This Fermilab Physics Class Library Task Force (ZOOM) would also expand the core of C++ expertise available for Fermilab physicists to draw upon.

C++ differs from Fortran in that the *potential* for common use of routines and libraries is greater. But this potential is not realized automatically. Unless coordination issues are considered from the start, utilities produced by one group generally do mot meet the needs of other groups—and each group ends up creating independant software.

To help increase code sharing, the centralized ZOOM task force must:

- Actively pursue outside (commercial and free-ware) packages. If ZOOM can verify that package X meets some needs in a sensible manner, then people can gravitate to that and not expend valuable development time.
- Act as a core for joint develpment of packages needed by both experiments.
- Develop relevant packages of sufficiently high quality as to overcome the natural reluctance of highly skilled physicists to rely on code developed by others. This means more extensive design thought and testing work than might be practical for some groups.
- Participate in cooperation with HEP groups outside the FNAL community, to acquire tools suitable for the Fermilab efforts. Of particular concern are areas where standardization is important, and thus a single product is more

valuable than two, even discounting any savings in effort. We must bring the ability to contribute some packages and the willingness to accept others from the HEP community.

ZOOM is answerable to the Run II Steering Committee, representing CDF, DØ, and the Computing Division. As implied above, the mission is to acquire, adapt, or (if necessary) develop modules that will be of use to both experiments. The products are organized into "packages" each of which contain—for a given platform—a library for linking with user code, and its sources (if not commercial) and build scripts. Multiple versions of the library may be present: for example, builds can be done with or without C++ exception handling enabled.

The ZOOM software, including sources and documentation, can be found on links from the homepage www.fnal.gov/docs/working-groups/fpcltf/fpcltf.html. In addition, directory trees containing source and binary libraries are kept on the DØ, CDF, and central Computing Division systems, so Run II users and others can link to the built libraries. ZOOM documentation is mostly html-based.

2 Foundations for the Libraries

2.1 Code/build Management

The ZOOM software is required to work on a specific set of compiler/system combinations designated by the Run II committee as being applicable for the Run II collaborations. The current platforms include KAI under SGI IRIX and under Linux (and soon under OSF/1), and Visual C++ on NT. Most ZOOM libraries also work with g++, but this is no longer a supported compiler and will not be until it catches up to the C++ standard. Similarly, vendor compilers will be added to the list of supported platforms as they come into sufficient compliance that the ZOOM code can work without distortion. Most vendors are moving rapidly in this direction.

Code and build management for ZOOM is via SoftRelTools (SRT), a CVS-based product also in use by CDF and DØ[1]. SRT is an HEP-wide product: Babar was a developer, and although the Babar version of SRT is distinct from the FNAL version, efforts are underway to reconcile those differences. ATLAS also uses SRT for its code management [2].

2.2 Portability

Because ZOOM intends to avoid the "Fortran in C++" style of coding, and to take full advantage of the benefits of C++, we take the following attitude when it comes to use of language features: We assume compilers support the ANSI C++ standard. Although we do not go out of our way to stretch the edges of the language, we will *strongly avoid* using inferior code design to accomodate deficiencies in specific

compilers. In cases where useful compilers do not meet the standard, we provide clean accomodations without distorting the intended code.

These accomodations form the basis of a portability strategy. For example, the standard libraries and headers may appear in non-standard ways on a platform; ZOOM provides links on such platforms so that user and ZOOM code will find the expected files at known places. The `ZMutility` package provides for this, for instance:

```
#include "ZMutility/iostream"
```

That package also provides various definitions and tricks to provide portability across all the supported platforms. This takes advantage of a mechanism in SRT called DEFECT defines. For example, on a given architecture, DEFECT_NO_VIRTUAL_COVARIANCE is checked, and if necessary defines are done so that the code will compile on systems that do not support covariant return types. The ZOOM code gets this portability by adding a single line:

```
#include "ZMenvironment.h"
```

User code can also take advantage of this, whether or not any other ZOOM packages are used.

The attitude toward not watering down our use of C++ features is modified in the case of packages adapted from or shared with outside HEP groups. Since commonality is sometimes of prime importance, and since others have already designed those packages, for such packages ZOOM adheres to their coding and language standards. Three examples: For HepTuple, we avoid any code that will cause fundmental problems with re-unifying our version with Babar's. ZOOM's version of CLHEP does not go beyond features in the original CLHEP code, so that ZOOM changes may safely be fed back. And the new C++ StdHep package will avoid language features that would limit its availablitly in the whole HEP community.

Non-Run II or non-SRT users (including those CDF or DØ members not using SRT or on not-yet-supported platforms) can still use ZOOM code. The code is kept in the CVS repository cvsuser@cdfsga.fnal.gov:/usr/people/cvsuser/repository. Mirrors of all code and documentaion are at /afs/fnal.gov/files/reports/workingroups/fpcltf/Pkg. A source code browser is available via the ZOOM home page.

2.3 Exceptions

It was immediately recognized that ZOOM ought to have a uniform way of dealing with anomalous situations. This mechanism had to deal with two needs: Some of our users disable C++ throw/catch exceptions, so ZOOM exceptions must have the option of relating to or avoiding C++ exceptions. And our user base is sophisticated enough to use hooks for associating handlers with particular exceptions, potentially dealing with and/or ignoring some set of anomalous conditions.

The ZOOM Exceptions package supports a hierarchical class structure of exceptions. Each ZOOM package—other than those intended for use outside the ZOOM environment, such as CLHEP—defines its tree of ZMexception-derived classes, and

copes with anomalous conditions by ZMthrow-ing the appropriate ZMexception. The user can hook pre-supplied or user-written handlers and loggers to particular exceptions (or subtrees of exceptions), and control whether an exception is ignored. If a ZMexception is *not* ignored, then either that object is thrown as a normal C++ exception or, if those are not enabled, the job is aborted.

The package provids a ZMerrno stack analogous to errno in Unix: The user code can examine the last (or the last-but-n) exception, find out what type it is and its position in the tree, look at its additional associated data, and so forth.

Although the Exceptions package was written for ZOOM packages to define and ZMthrow ZMexceptions, users can and have used the same package to define their own custom exception hierarchies.

3 The ZOOM Packages

The current ZOOM packages are:

CLHEP, a uniform version for Run II users. This has eliminated the situation of several CLHEP versions floating around CDF and DØ, and provides a version known to compile on all our supported platforms and a means of being responsive to Run II needs. ZOOM CLHEP is "stand alone"—no features specific to the ZOOM environmend or additional language constructs are used. It is fully usable wherever CLHEP can be used.

We wish to avoid divergent versions of CLHEP. To this end, *all* ZOOM modificatiosn are fed back as proposed changes, and the CLHEP editors' comments are heeded. To make feedback smooth, all enhancements other than true bug fixes respect backward compatibility with our snapshot of the original CLHEP behavior.

ErrorLogger is a common framework for message logging and error statistics.

HepTuple deals with histograms and n-tuples. This package is intended for joint use by Run II and Babar. The interface is on top of a choice of "manager." Histograms and n-tuples can be accumulated and read back. HepTuple features are a superset of those in HBOOK, which may be used as the manager. Another option is to use the "lighter-weight" Histoscope manager, which does not require ZEBRA. Histoscope has been augmented: Column-wise and disk-resident n-tuples are now supported, the read-back speed is much greater than for HBOOK, and an enhanced column-wise n-tuple browser is being developed.

LinearAlgebra deals with matrices, decomposition and solving, eigen-analysis, etc. It has a coherent way of treating specialized matrices, and is low overhead for small-matrix computations. The core of this C++ has been in use at the FNAL Accelerator division for quite a few years.

SIunits [3] provides dimension and units checking for physical calculations, at no runtime cost in time or memory. It also provides a comprehensive list of constants of nature.

PhysicsVectors deals with 3-vectors, 4-vectors, rotations, and Lorentz transformations. In comparison to the CLHEP Vector sub-package it has the concept of different coordinate views of a vector (e.g. `v.phi() = PI/3`), pseudorapiditiy as a substitute for θ, all standard forms for rotations including Euler angles, specialized classes such as LorentzBoost for efficiency, and a number of HEP methods that our collider users wanted such as deltaR(). It also has been subjected to an extensive testing suite.

ZMtools is a grab-bag of portable tools. It currently has a portable ZMtimer class, and ZMspline will be added soon.

ZMutility and **Exceptions** are foundations for all packages, described above.

4 CLHEP and ZOOM

Each CLHEP sub-package is either "validated" or merely "present" in in the ZOOM version. Present packages are vetted for compilation on our supported platforms, but ZOOM does not make other changes, and does not look at the quality of the package. Validating a sub-package means ZOOM takes repsonsibility for testing for proper behavior, and adding enhancements our users want. It also commits that ZOOM will not later produce or recommmend a different product for the same niche as this CLHEP subproduct.

ZOOM is tightly involved with two sub-packages of CLHEP: Random, where we are validating the existing package, and StdHep, which we hope to contribute as a new package to CLHEP.

4.1 CLHEP Random

The ZOOM-validated CLHEP Random has general enhancements including a few feature additions requested by CDF and DØ users, minor bug fixes, the usual CLHEP porting to Run II platforms, and semantics extensions that are natural to C++. For example, constructors for HepRandom distributions now accept arguments representing default values for the parameters of the distribution.

We have expanded the set of distributions to include all those listed in table 28.1 of *Review of Particle Physics* [4], adding Binomial, Poisson, χ-squared, Student's t, and the Gamma distribution. (We will be adding Multivariate Gaussian, which appears in the new edition.) Also, an efficient form of Landau, Vavilov, and spin-1/2 Vavilov distributions will be added; the Vavilov distribution should be much faster than the CERNLIB version.

We have included a number of additional random engines. These include DualRand, used on ACPMAPS and in Edinburgh for Lattice QCD; RandHurd, a fast generator based on interconnected shift registers; Ranshi, a huge-seed engine based on simulating a collection of spinning balls; Mersenne Twister which is a fast huge-seed method with excellent mathematical grounding; and a double-precision version

of the familiar Ranlux engine. Time performance of these CLHEP engines has been evaluated by Sverre Jarp [5].

Finally, we have run the Marsaglia DIEHARD test suite against each engine, posting the results on the web, to present data on their randomness properties are. Still to come is independant verification that each distribution class gives the correct distribution.

4.2 StdHep

The StdHep package is meant as an implementation of a standard way to describe particle events. It incorporates the system of assigning ID numbers to each particle type in section 31 of [4], and the HEPEVT standardized event structure. It also deals with interchange of data among the formats used by the popular event generators (e.g., ISAJET, Pythia, QQ, Herwig, DPMJET, Jetset, ...).

Here, the crucial aspect is that the HEP community needs one (and not two) such standards. So it is obviously right to do this as part of CLHEP. Fortunately, a Fortran implementation exists; the documentation for this[6] defines and limits the features needed in the C++ version. These include utilities for xdr I/O, event display, transitions to various formats, particle properties and decay routines, tree traversal of the decay chain, and boosts and rotations.

The principal implementor of the latest version of the Fortran package will be the main coder of the C++ package. The intent is to provide clean design and natural semantics, taking advantage of C++ constructs. We also intend to consider the issues of working in conjunction with newer C++ packages such as Geant4.

The aim is to feed StdHep into CLHEP for general HEP use.

References

[1] *Software Management at Fermilab* by Rob Harris, parallel talk 30, this conference.

[2] *Overview of ATLAS Software Release Tools* by Lassi Tuura, parallel talk 188, this conference.

[3] *he SI Library of Unit-Based Computation*, by Walter Brown, parallel talk 148, this conference.

[4] *Review of Particle Physics*, European Physical Journal C3, 1998

[5] *Performance of C++ Random Number Generators* by Sverre Jarp, parallel talk 24, this conference.

[6] *StdHep 4.02 Monte Carlo Standardization at FANL* by Lynn Garren, Fermilab PM0091, http://www-pat.fnal.gov/stdhep.html