Histograms as Objects

Y. Adesanya

IT Division, CERN, CH-1211 Geneva 23, Switzerland

In the survey of software needed for computing in the LHC era [1], we did not find software appropriate for our needs in the area of histogramming. We started a project to develop a histogramming package. Requirements gathered from several users, governed the analysis and design of the package. Among the requirements were support for arbitrary dimension of histograms and profile histograms, filling of various types of data (e.g. not only real numbers), correct handling of statistics, extendable design and a coherent user interface. Ntuples were explicitly excluded from the package. Persistent storage of histograms shall be handled via an object database.
The analysis resulted in a separation of the binning process: The identification of a data point's associated bin is distinct from the accumulation of data points in the histogram. Thus the histogram consists of an object cumulating the data points according to a partition object for each coordinate. Various histograms may share the same partition objects. The implementation in C++ is in progress. Objects created can be stored in the Objectivity ODBMS [2]. Modules to display histograms in the framework of IRIS Explorer [3] are being developed in parallel. A first version has been given to a few users.

Key words: Analysis, C++, Class libraries, Data presentation

1 Designing histograms for HEP

Before the decision was taken to develop HEP histograms, some individuals had already contemplated creating their own ad-hoc classes. It was time for a common solution. An evaluation of Rogue Waves Math.h++ histogram class showed several limitations. Work began on formalizing histogram package requirements. The criteria includes:

- Support for weighted entries
- Multi-dimensional histograms

– Recording underflow and overflow
– Variety of binning systems which can be extended
– Persistence capability
– General Transformations
– Statistics

Once a draft requirements document had been produced, an initial class diagram was derived, see fig. (1). Its most striking design feature is the separation of the problem space (Partition), where the element type of the binning system and its bin boundaries are defined. *Histogram* objects contain a collection of bins which store weights and errors. *Partition* instances are responsible for generating a bin identifier from a data point.

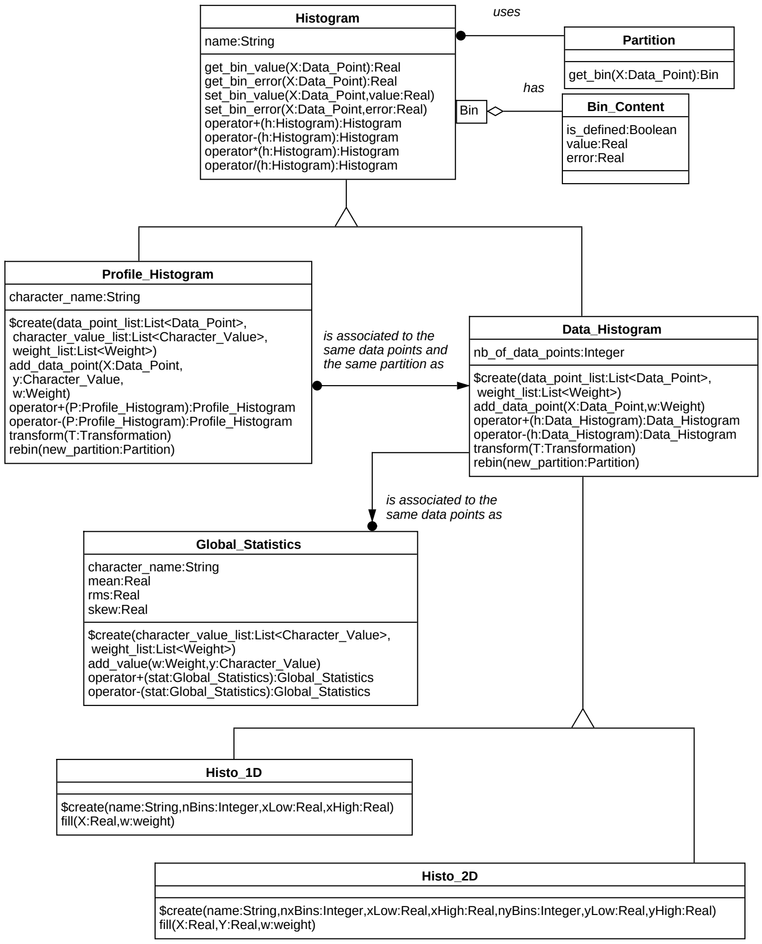

Fig. 1. A Top-level OMT diagram of the Histogram classes

The classes are connected via a one-to-many relationship. This acknowledges the fact that users often wish to create several histograms with the same characteristics in a single program. Bin contents can be set through the *Histogram* member functions:

– set_bin_value(X:Data_point,value:Real)

– set_bin_error(X:Data_point,error:Real)

Both calls cause the recipient *Histogram* to retrieve the bin qualifier ID from its respective partition. The appropriate bin can then be referenced and the value or error is returned. Histograms can also undergo operations such as addition, multiplication and subtraction. The OMT diagram lists operations which accept *Histogram* objects as arguments. This will be extended to scalar types in the future. Two subclasses directly inherit from the *Histogram* class: *DataHistogram* is for accumulating weights. The other subclass is *ProfileHistogram*. The *Partition* class holds a list of references to elementary partitions. An *ElementaryPartition* object is abstract, corresponding to a single dimension within a partition. Through sub-classing, different kinds of dimension are implemented such as fixed length, variable length, sets, etc. *ElementaryPartition* objects can also be shared amongst different *Partition* objects. The design is fairly open in terms of extensibility. The initial class structure had been defined but concrete test cases would aid validation and refinement. Prototypes would act as a catalyst by promoting user feedback. Other issues included the interface design. Storage also had to be dealt with. The RD45 [4] group provided assistance in the production of persistent-cable code, recommending Objectivity as the choice of ODBMS.

2 Creation and access of persistent histograms

A subset of the histogram class structure was chosen for immediate coding, enabling the creation of data histograms of n-dimension (fixed/variable width bins). The base types of the data members belonging to persistent capable classes were switched to ODMG (Object Database Management Group) compliant types. These types have a fixed implementation to overcome compiler and machine architecture differences. C++ memory pointers cannot be stored inside persistent classes. Objectivity provides a set of generic "smart" pointers used to reference persistent objects. The de-reference operators are overloaded so object data members and functions can be accessed just like any normal object pointer. Objectivity also provides persistent string classes and STL-like [5] containers including vectors and ordered lists which can hold persistent objects or their references.

The histogram class requirements deliberately steer clear of visualization details. In reality, no vendor supplies a shrink-wrapped HEP visualization system ready to ship. An MVS (Modular Visualization System) such as IRIS Explorer is designed to allow a high degree of extensibility through code integration. A HEP tailored visualization package would include function fitting and minimalization through a MINUIT interface as well as tools to manipulate the persistent histograms. Typically, an off-the-shelf MVS environment will consist of general purpose modules such as:

- File readers/writers
- Filters for extracting data subsets
- Transformers which apply functions to datasets
- 2-D and 3-D graphics toolkits

New modules can be built with existing code using an API provided by the MVS. Data is shared between modules through a variety of types designed to handle different forms of visualization information, from datasets to graphics geometry. There is also the possibility to create user-defined types: The protoyping objectives were:

- Read in histograms
- Provide an interface to operations
- Render the histogram data

A suite of histogram modules have been written for IRIS Explorer, as part of a HEP-oriented visualisation environment (HEP Explorer). They allow persistent histograms to read in from an Objectivity database and minimalisation can be performed via an interactive interface to the MINUIT package [6]. Type-checked operations can be performed and the data can be visualised using traditional 2D graphs, contours as well as 3D lego plots and isosurfaces. The prototype histogram library will continue to be extended in parallel with the analysis/design documententation, with the aid of feedback from histogram users including CMS, NA45 and the GEANT team.

References

[1] The LHC++ group, *Libraries for HEP Computing - LHC++* (Found at http://wwwcn.cern.ch/asd/lhc++.html).

[2] Objectivity inc., *Choosing an ODBMS*(Objectivity,California,1996).

[3] The Numerical Algorithms Group, *IRIS Explorer Center*
(Found at http://www.nag.co.uk/Welcome_IEC.html).

[4] The RD45 group, *RD45 - A Persistent Object Manager for HEP*(CERN/LHCC, Geneva,1996).

[5] M. Lee, A. Stepanov, *The Standard Template Library* (Hewlett-Packard, California,1995).

[6] Pierre Lecointe, *The IRIS Explorer Data Fit Module*
(Found at http://sgrabber.cern.ch/~lecointe/)