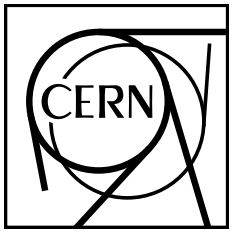

CERN Computer Program Library Documentation

Histogram Template Library

User Guide

Application Software and Databases Group

Information Technologies Division

CERN Geneva, Switzerland

The source of this document is marked up in XML using a DTD copied on a subset of LaTeX's functionality. It is translated with an XSL style sheet and James Clark's `xt` Java program into LaTeX and HTML. The LaTeX source is further typeset using the `cernman` class file developed at CERN and a printable PostScript file is generated.

Acknowledgements

HTL has benefited from the suggestions, advice and help of many individuals. In particular the major part of HTL was written during 1998 by Savrak Sar, who worked for sixteen months at CERN as a French *coopérant.*
A special mention should be given to Yemi Adesanya and Jacub Moscicki for the existing HistOOgrams package and the templated prototype; Dirk Düllman and Marcin Nowak for help with Objectivity; Vincenzo Innocente for his version of templated histograms; Olivier Couet and Michel Goossens for general support.

Contact Person: Dino Ferrero Merlino /IT `Bernardino.ferrero.merlino@cern.ch`
Documentation: Michel Goossens /IT `michel.goossens@cern.ch`

Edition – April 1999 Printed May 5, 1999

Table of Contents

Part I

Getting started

Chapter 1: Overview

1.1 What is HTL?

The Histogram Template Library (HTL) is a C++ class library that provides powerful histogramming functionality. As the name suggests, it exploits the *template* facility of C++ and is designed to be compact, extensible, modular and performant. As such it only deals with histograms - i.e. binned data - and not unbinned or "Ntuple" data. Furthermore, although simple file-based I/O and "lineprinter" output are supported, it is decoupled from more advanced I/O and visualisation techniques. In the context of LHC++, such capabilities are provided by other components that fully interoperate with HTL.

In the High Energy Physics (HEP) community, the HBOOK package has been the *de-facto* standard for histogramming for many years. This manual is written assuming a basic familiarity with HBOOK (see Table 1.1).

HTL itself offers the histogramming features of HBOOK as well as a number of useful extensions, with an object-oriented (O-O) approach. These features include the following:

- booking and filling of 1D, 2D and profile histograms;
- computation of statistics such as the mean or rms of a histogram;
- support for operations between histograms;
- browsing of and access to characteristics of individual histograms.

This package replaces the histOOgrams package - an earlier C++ class library for histograms. The major functional innovation over the previous package are the support for different kinds of bins, the support of both persistent and transient (i.e. in-memory) histograms at runtime and the definition of an abstract histogram interface.

As a result, it is now possible to work with transient histograms and subsequently save some or all of them in a database as persistent histograms in a very simple and natural way (thus simulating so called explicit I/O). This clearly has significant performance advantages, particularly in the area of filling operations.

It is also possible to work with transient histograms only. These may be printed or stored in files - the latter using a simple ASCII table format that can be readily used by widely available tools, including PAW or Excel.

The definition of an abstract histogram interface allows functionality that is provided by external packages, such as plotting or fitting, to be decoupled from the actual implementation of the histogram. This feature paves the way for co-existence of different histogram packages that conform to the abstract interface.

Table 1.1: HBOOK routines and their HTL equivalent

HBOOK name	*HTL equivalent*
HBOOK1	`Histo1D`
HBOOK2	`Histo2D`
HFILL	`fill` method
HPRINT	`print` method

1.2 Use of templates

Although HTL uses templates internally, a small set of ‘user-classes’ are provided. These have a very familiar (i.e. HBOOK-like) look-and-feel and eliminate the need for users to be exposed to templates.

Nevertheless, the package still requires adequate template support at the C++ compiler level. More details are given in Appendix A.

1.3 Motivation

When the existing HistOOgram package was implemented, compilers that provided adequate support for templates were not widely available. Templates and other modern C++ features can offer significant advantages. In the case of HTL, these can be summarised as:

- increased performance;
- greater flexibility and maintainability;
- the support of both persistent and transient histograms at runtime. Both provide similar features and are generated from the same source files;
- a more O-O approach, which in turn results in better encapsulation and more decoupling.

These issues, especially the maintainability and high degree of decoupling, are of particular importance to very long-term projects, such as those at the LHC, where change and migration must be assumed and planned for.

Chapter 2: HTL at a glance

This chapter provides an overview of HTL features. Example code is based on the use of the non-templated classes that are recommended for use in typical analysis jobs. The HTL package itself includes a complete set of working examples that can be used as a basis for building more complex applications. A description of these examples can be found in Appendix D.

2.1 Booking and filling persistent histograms

The basic operations of creating (booking) and filling histograms are shown below. As the code fragments illustrate, these operations are performed using methods that have the same basic signature as the equivalent HBOOK subroutines. The primary difference is that the HBOOK histogram identifier (ID) is replaced by "smart pointers". These pointers indicate both the type of the histogram to which they refer (e.g. 1D, 2D) and whether they are transient or not.

As will be familiar to all HBOOK users, the following information is specified at booking time:

- A character string or *title*, normally printed when the histogram is displayed.
- The number of bins.
- The lower bound of the histogram axis.
- The upper bound of the histogram axis.

For histograms of more than one dimension, the last three arguments are then repeated for each dimension.

The following example shows the booking and filling of a 1D histogram of 20 bins and a 2D histogram of 50 bins in both X and Y axes.

```
#include "HTL/PHistograms.h" // Persistent histograms.
...
// Booking a 20 bin 1D histogram
 HepRef(PHisto1D) histo1 =
  new (db_) PHisto1D("A 1D histogram",20,0.,40.);
// Booking a 50X50 bins 2D histogram
 HepRef(PHisto2D) histo2 =
  new (db_) PHisto2D("A 2D histogram",50,5.,15.,50,5.,15.);

// Filling
 double x, w = 0.5;
 for( long i=0; i<50000; i++ ) {
  x = -i*sin(float(i));
  histo->fill(x,w);
  histo2->fill(x,x,w);
 }
```

The current implementation of HTL uses an Object Database (ODBMS) to provide powerful and scalable persistence capabilities that go beyond what is possible with simple files. [1]

In the following, it is assumed that the reader is familiar with the basic concepts of using an ODBMS. For example, any operations on persistent objects must be performed in the context of a *transaction* and

[1] Simple file-based persistence is also provided - see chapter 2 for a description of the *write* method.

a database session must have been established. The necessary code to perform such operations has been omitted in the interests of clarity.
Although the basic histogram operations that are performed in the above example will be clear to HBOOK users, the use of "smart-pointers" differs from the traditional use of histogram IDs and needs further explanation. Consider the following code fragment:

```
HepRef(PHisto1D) histo1 = new ...
```

rather than

```
PHisto1D *histo1 = new ...
```

In this piece of code, `HepRef` is a smart pointer to a persistent histogram. As the previous example shows, a smart pointer can be used as a standard C++ pointer: that is, you can dereference it in the normal C++ manner:

```
histo1->fill(...)
```

In addition to the use of smart pointers, the `new()` operator should be studied. It is followed by a so-called *clustering hint*. A clustering hint is used to indicate where the persistent objects - histograms in this case - should be physically located on persistent storage. The database attempts to place new objects "close" to those referenced by the hint. In general, the efficient use of clustering allows performance to be maximised, as objects are transferred to and from disk and across the network in database pages. Effective clustering minimises unnecessary I/O and improves cache efficiency.
Finally, we note that the histograms created with `new()` are not deleted with a corresponding `delete()` operator. This would not only delete the histogram in the running application *but also* from persistent storage, which is presumably not the intention.

2.2 Booking and filling transient histograms

The previous example is now repeated for the case of transient histograms. Once again, two histograms are created and filled. Largely speaking, the code required is identical to the persistent case. The differences between the two are discussed in futher detail below.

```
#include "HTL/Histograms.h" // Transient histograms.
...
// Booking a 20 bin 1D histogram
 Histo1D *histo1 = new Histo1D("A 1D histogram",20,0.,40.);
// Booking a 50X50 bins 2D histogram
 Histo2D *histo2 = new Histo2D("A 2D histogram",50,5.,15.,50,5.,15.);

// Filling
 double x, w = 0.5;
 for( long i=0; i<50000; i++ ) {
  x = -i*sin(float(i));
  histo->fill(x,w);
  histo2->fill(x,x,w);
...
delete histo1;
delete histo2;
```

Transient histograms are normal, in-memory C++ objects so standard C++ pointers are used.
We note that the `new()` operator is now the standard C++ one. In this case, it is necessary to also handle explicit deletion of the histogram to avoid memory leaks.
In summary, the differences between transient and persistent HTL usage requires only the selection of the corresponding class names (`Histo` versus `PHisto`) and the use of smart pointer in the persistent case.

2.3 Making a transient histogram persistent

As already mentioned, HTL makes it possible to use both persistent and transient histograms at runtime in the same program.

Since operations on transient histograms are significantly faster than those on persistent ones, a typical procedure would be as follows:

1. perform all of the desired operations on transient histograms (filling in particular)
2. make persistent copies of these transient histograms.

This offers the “best of both worlds” - the performance of the transient implementation plus the added value of persistence.

To make a transient histogram `ph` persistent:

1. declare a persistent histogram `ph` analogous to the transient counterpart,
2. use the `new` operator as for the booking and then
3. pass the histo interface of `ph` to the constructor of `ph`.

A persistent copy of the transient histogram now exists. These steps are shown in the following code fragment.

```
#include "HTL/Histograms.h" // Transient histograms.
#include "HTL/PHistograms.h" // Persistent histograms.

// Declare and define ‘th’:
 Histo1D  *th = new Histo1D( "Mass",20,0.,4000.);

// Process ‘th’, perform operations:

// Copy ‘th’ to the database, i.e. make ‘th’ persistent:
//
HepRef( PHisto1D ) ph = new (dbH ) PHisto1D( *th );
// This create a persistent histo that is a clone (with data) of transient histo ‘th’.
```

We note that it is possible to create a persistent histogram out of *any* histogram interface, be the source histogram transient or persistent itself.

2.4 HTL histogram classes

As described above, HTL is based on template classes which allow very different types of histogram to be supported using the smallest amount of code.

Nevertheless, the emphasis on usability has been maintained and hence a small set of simple, easy-to-use simple HBOOK-like classes are provided. It is expected that these classes will cover most cases. The list of available classes is given below.

Note that all of these classes are used in the same way. For example, to book a histogram of a given type one uses:

```
// Use HepRef to refer to persistent classes, C++ pointers for transient ones

 HepRef(PHistyyy) histo1 =   new (dbH) PHistoyyy( ...
 Histyyy *histo1 =   new  Histoyyy( ...

// Use Histo1D for 1D Histogram, Histo2D for 2D Histogram,
// Histo2DVar for 2D histogram with variable binning etc.
```

e.g., to repeat the initial example:

```
Histo1D *histo1 = new Histo1D("A 1D transient histogram",20,0.,40.);
```

2.4.1 Persistent classes

`PHisto1D`	1D histogram with `Gravity_Bin_1D` and fixed binning;
`PHisto1DVar`	1D histogram with `Gravity_Bin_1D` and variable binning;
`PProfileHisto`	1D histogram with `Profile_Bin` and fixed binning;
`PProfileHistoVar`	1D histogram with `Profile_Bin` and variable binning;
`PHisto2D`	2D histogram with `Weighted_Bin` and fixed binning;
`PHisto2DF`	2D histogram with `Float_Weighted_Bin` and fixed binning;
`PHisto2DVar`	2D histogram with `Weighted_Bin` and variable binning.

All classes store bin values and errors as *doubles*, except `PHisto2DF`, which stores them as *floats* (thereby minimising the storage requirements for very large 2D histograms).

Weighted bins are equivalent to HBOOK bins, Gravity bins store the position of the bin center-of-mass, so that the position of the bin center is more accurate when the bin count is low or the binning is not optimal (at least in some cases). More details on bin types will be given later.

Notice that profile histograms and 2D histograms with variable binning were not available in HBOOK.

2.4.2 Transient classes

`Histo1D`	1D histogram with `Gravity_Bin_1D` and fixed binning;
`Histo1DVar`	1D histogram with `Gravity_Bin_1D` and variable binning;
`ProfileHisto`	1D histogram with `Profile_Bin` and fixed binning;
`ProfileHistoVar`	1D histogram with `Profile_Bin` and variable binning;
`Histo2D`	2D histogram with `Weighted_Bin` and fixed binning;
`Histo2DF`	2D histogram with `Float_Weighted_Bin` and fixed binning;
`Histo2DVar`	2D histogram with `Weighted_Bin` and variable binning.

All notes relating to persistent classes are also valid for transient classes.

2.5 Accessing bin content, bin error, bin center

A histogram stores its bins and allows to access their content, error and center.

`h1D->bin(i).value();` Value of the bin indexed by "`i`".

`h1D->bin(i).error();` Error of the bin indexed by "`i`".

`h1D->bin_center(i);` center of bin index by "`i`".

Notice that the bin object can retrieve its content/error but not its center; bins are simple containers, only the histogram knows where they are mapped in the binning space.

2.6 Line-printer and ASCII table output

Printing a histogram in text mode is required in a number of cases. For example, it can be useful for long batch jobs, where the user is mainly interested in checking the quality of data.

HTL provides a text print helper for this purpose, that is largely reused from the previous HistOOgrams packages.

The following example shows how the `print` method can be invoked to generate a simple line-printer style representation of a histogram.

```
#include "HTL/Histograms.h" // Transient histograms.

Histo1D *histot = new Histo1D("Transient Histo_1D", 20, 0.0, 20.0 );
HepRef(PHisto2D) histop =
  new PHisto2D("Persistent Histo_2D", 20, 0.0, 20.0, 20, 0.0, 20.0 );
...
HPrinter hp( cout );
hp.print( *histop );
hp.print( *histot );
delete histot;
```

The above example demonstrates the use of the abstract histogram interface: the same helper object can print any histogram that conforms to the interface, regardless of whether it is transient, persistent, 1D, 2D and so forth.

Particularly when working with transient HTL, it can be convenient to dump a histogram in a simple ASCII table that can then be read by standard tools (PAW, Excel, IRIS Explorer...) to plot or fit it.

```
#include "HTL/Histograms.h" // Transient histograms.

T_Histo1D *histo = new Histo1D("Histo_1D", 20, 0.0, 20.0 );
...
HistoTable1D ht1 ("histo.txt");
ht1.write(*histo);
delete histo;
```

Chapter 3: Partitions and Bins

3.1 Partitions

The concept of a "Partition" was first introduced in the HistOOgrams package. A partition describes how one dimension of the problem space is divided into intervals. Consider the case of a 50 bin histogram in the range $[0, 50]$, a partition object containing the number of bins and the interval limits will describe completely how we divide such interval: a set of 50 sub-intervals of equal width. This is termed a "Fixed" or "Even" partition.

It may be required to book an histogram over the same range as the example above, but with bins of variable widths. In this case, a partition containing the number of bins, the lower limit of each sub-interval and the upper limit of the last sub-interval will describe completely $[0, 50]$ interval is divided. Such a partition is termed a "variable" or "Uneven" partition.

An *n-dimensional* HTL histogram thus contains *n* partitions, one for each axis. The only concern of a partition is to associate any ordered 1D space with a discreet numbered space. Thus it associates an interval to a positive integer. Hence, a partition knows about the width of the intervals and their lower point/bound or upper point/bound.

In the HTL package, a histogram delegates to its partitions the task of locating a bin. In other words, information about the lower and upper bounds of a bin or the width of a given bin are obtained from the corresponding partition. This is shown in the following code fragment, which demonstrates how the lower and upper bound and width of a given bin can be obtained.

```
Histo1D *histo = new Histo1D("Transient Histo_1D", 20, 0.0, 20.0 );
...
histo->partition().lower_point(i) // Obtain the lower bound of bin i
histo->partition().bin_width(i)   // and its width
histo->partition().upper_point(i) // and upper bound
```

3.2 Bins

Bins themselves contain information about the content, the error and possibly the center of the bin. Bin information is always accessed through interface methods. HTL provides the following types of bin:

`Weighted_Bin` class representing bins with weighted data points. By default the gravity center of these bins are the middle or center of the bin.

`Gravity_Bin_1D` class representing bins with weighted data points and that know how to determine the gravity center of the bins. This new bin class allows us to have filling time statistics as in the existing histogram package, but without having to compute separate quantities at filling time. The other advantage is that in case of bins with low counting, the information of the bin center is more accurate that the middle of the bin.

Also, `Gravity_Bins` could provide more accurate information in some cases of non-optimal binning. For instance, if you sample an integer variable in the range $[0, 20]$ with a 20 bin histogram over the same interval, the bin center is properly computed at the integer values, while in a traditional histograms they would correspond to the closest half integer.

`Profile_Bin` class representing bins that can average another quantity.

Note that bins only know about their contents. They do *not* know where they are are located in the histogram to which they belong, nor about their widths or bounds - this information is stored in the partition to which they belong, which also defines the bin layout within a histogram.

Chapter 4: Histograms with variable partitions and Profile histograms

4.1 Variable partition histograms

HTL supports 1D and 2D histograms with variable partitions. The partitions are specified as a vector of real numbers.

4.1.1 Persistent case

```
P_Points_Vector points_on_X(10); // Vector of 10 real numbers
points_on_X[0] = 0.0; points_on_X[1] = 1.0;
points_on_X[2] = 3.5; points_on_X [3] = 4.0;
points_on_X[4] = 5.5; points_on_X[5] = 9.0;
points_on_X[6] = 10.; points_on_X[7] = 15.;
points_on_X[8] = 19.; points_on_X[9] = 20.;

HepRef(PHisto1DVar) histo =
  new (db_) PHisto1DVar( "Histo1DVar", points_on_X );

P_Points_Vector points_on_Y(10);
points_on_Y[0] = 0.0; points_on_Y[1] = 1.0;
points_on_Y[2] = 2.5; points_on_Y[3] = 4.0;
points_on_Y[4] = 6.0; points_on_Y[5] = 9.0;
points_on_Y[6] = 12.; points_on_Y[7] = 14.;
points_on_Y[8] = 18.; points_on_Y[9] = 20.;

HepRef(PHisto2DVar) histo =
  new (db_) PHisto2DVar( "Histo2DVar", points_on_X, points_on_Y );
```

4.1.2 Transient case

```
T_Points_Vector points_on_X(10); // Vector of 10 real numbers
points_on_X[0] = 0.0; points_on_X[1] = 1.0;
points_on_X[2] = 3.5; points_on_X[3] = 4.0;
points_on_X[4] = 5.5; points_on_X[5] = 9.0;
points_on_X[6] = 10.; points_on_X[7] = 15.;
points_on_X[8] = 19.; points_on_X[9] = 20.;

Histo1DVar *histo = new Histo1DVar( "Histo1DVar", points_on_X );

T_Points_Vector points_on_Y(10);
points_on_Y[0] = 0.0; points_on_Y[1] = 1.0;
points_on_Y[2] = 2.5; points_on_Y[3] = 4.0;
points_on_Y[4] = 6.0; points_on_Y[5] = 9.0;
points_on_Y[6] = 12.; points_on_Y[7] = 14.;
points_on_Y[8] = 18.; points_on_Y[9] = 20.;

Histo2DVar *histo2 = new Histo2DVar( "Histo2DVar", points_on_X, points_on_Y );
```

Both the class names and type of vector used depends on whether transient or persistent histograms are involved.

4.2 Profile Histograms

Profile histograms are implemented both with fixed and variable partitions. The bin errors are computed as in HBOOK, but only the `' '` and `'S'` options are supported (see the HBOOK manual for a definition of these options and the associated error calculation). However, unlike in HBOOK these are not a booking time options; you can ask each bin to compute either the Spread or the Spread/sqrt(N) error at run-time.

```
HepRef(PProfileHisto) histo = new (db_) PProfileHisto("Profile",20,0.,1.);
long i;
double x;
for( i=0; i<50000; i++ ) {
  x = drand48();
  histo->fill(x,x,1.);
}

for( i=0; i<histo->bin_count(); i++ ) {
  cout << endl << ". " << setw(2) << i << ": "
<< setw(9) << histo->i_bin(i).value() << " Error "
<< setw(4) << histo->i_bin(i).error(0) << " Spread Error "
<< setw(4) << histo->i_bin(i).error(1);
}
```

As you can see the `error()` method on the bin interface allows to access a set of errors: in this case the `Profile_Bin` class accepts either the 0 or 1 index, which map respectively to Spread/sqrt(N) and Spread errors.

Profile histograms can be filled with weights, but use of negative weights yield meaningless results. With weights different from 1. the errors are calculated properly only for bins with non-zero spread, and set to 0. for bins with zero spread (HBOOK-like).

Chapter 5: Operations on histograms

5.1 What do histogram operations apply to?

In the HBOOK package, operations on histograms result in the creation of a new histogram, rather than working directly on the target histogram. This is not an approach which fits naturally in the Object Oriented world of C++, where responsibility for object deletion and creation should normally go together.

In HTL, one first creates a copy of the target histogram and then applies the operation on the newly created copy. The responsibility for the creation and deletion of the new histogram are thus delegated to the target application and unnoticed memory leaks potentially avoided.

5.2 Cloning a histogram

HTL allows you to produce a new copy of an existing histogram using the `clone` method of the `I_Histogram` interface. The first argument, if given, specifies where the histogram should be cloned, that is the clustering hint. By default the clustering hint is the container that contains the source histogram in the persistent case and empty in the transient case. The last argument, if any, specifies how the histogram should be cloned:

- Specifying a value of 0 will result in a clone of just the structure;
- A non-zero argument, which is also the default, will cause the data to be copied in addition.

An example of cloning in the persistent and transient case follows.

```
* Persistent case */

HepRef(PHisto1D) h1D =
  new(db_)  PHisto1D( "Histo1: parabolic function", 20,0.,20.);
...
// The persistent histogram will be cloned close to the original
// object (default behaviour)
HepRef(PHisto1D) clone_1D = h1D->clone();   // Clone 'h1D' structure and data

/* Transient case */

Histo1D *h1D = new Histo1D( "Histo1: parabolic function", 20,0.,20.);
...
Histo1D *clone_1D = h1D->clone();   // Clone 'h1D' structure and data
```

5.3 Operations on histograms

Below is a list of methods implementing operations on histograms.

5.3.1 Operations with scalars

`void add( double x )` Add "x" to current histo.

`void sub( double x )` Substract "x" from current histo.

`void mul( double x )` Multiply "x" with current histo.

`void div( double x )` Divide current histo by "x".

5.3.2 Operations with another histogram

`Ref_Like_Current add( const Ref_Like_Current &other )` Add "`other`" histo to current histo.

`void add( const I_Histo &other )`

`Ref_Like_Current sub( const Ref_Like_Current &other )` Subtract "`other`" histo from current histo.

`void sub( const I_Histo &other )`

`Ref_Like_Current mul( const Ref_Like_Current &other )` Multiply "`other`" histo with current histo.

`void mul( const I_Histo &other )`

`Ref_Like_Current div( const Ref_Like_Current &other )` Divide current histogram by "`other`" histo.

`void div( const I_Histo &other )`

`Ref_Like_Current binomial_div( const Ref_Like_Current &other )` Divide current histo by "`other`" histo using "binomial error"

`void binomial_div( const I_Histo &other )`

Notice that operations with another histogram exists with two signatures, the first one allowing to directly assign the result of an operation.

This is an example of code to add two histograms:

```
HepRef(PHisto1D) histo1 =
  new(db_)  PHisto1D( "Histo1: parabolic function", 20,0.,20.);
  ...
  for (int i = 0; i < 50000; i++)
    histo1->fill(x,w);

  // Now create a clone of histo1
  HepRef(Histo1D) histo2 = histo1->clone();
  // Add histo1 to histo2 and change histogram name
  histo2->add( *histo1 );
  histo2->set_name( "Histo2 =  Histo2+Histo1 = 2*Histo1" );

  //
  // It is also possible to clone and perform an operation on single
  // statement:
  //         histo2 = histo1->clone()->add( histo1 );
  //
```

As discussed in the first paragraph, it's important to first make a clone of the original histogram. Another way to clone a histogram is using a copy constructor, as described in Chapter 8.

Part II

Advanced HTL

Chapter 6: Histogram Factories and Naming Trees

This section describes a number of techniques for handling the naming and location of persistent histograms in the database. As such, this section only applies to persistent HTL histograms as they

6.1 Histogram factories

The Object Database Management Group (ODMG) standard for Object Databases specifies that persistent objects are created using a `new()` operator with a *clustering hint* argument. This parameter provides a mechanism whereby the user or application code can specify where each individual object is physically stored. This is demonstrated by the the following code fragment, where the `histo1` object is created in the database referred to by the database handle `db_`.

```
HepRef(PHisto1D) histo1 =
  new (db_) PHisto1D("A 1D histogram",20,0.,40.);
```

Not only is this syntax somewhat unusual to novice users, but also it poses problems when attempting to use tools such as the SWIG interface generator for scripting languages. [1]

The `HFactory` class overcomes this problem by letting the user specify by name the database or container in which histograms should be created and provides a factory method for the actual creation of the histograms.

The following code fragment demonstrates the usage of a Histogram Factory. It first initialises the factory, specifying the target database and container by name. Histograms are then created and filled and are automatically stored in the appropriate container.

```
#include "HTL/H_Factory.h"
...
  HFactory myFactory;
  // Create histograms in DataBase "gepo", Container "sbaffini"
  if (myFactory.init(this,"gepo","sbaffini")) {
    // Create a histogram using the factory
    HepRef(PHisto1D) h1 = myFactory.Histo1D("Histo-1",noBins,0.0,20.0);
   ...
   h1->fill(x, 1.);
```

The `HFactory` class provides a method to generate all useful types of histograms. The factory methods are named according to the histogram type and take parameters according to the corresponding constructor.

```
int init (HepDbApplication *sess, char *dbname,  char *contname = 0);
HepRef(PHisto1D) Histo1D (const char *a_title, Size n, double x1,
              double x2, End_Point_Convention epc = RIGHT_OPEN);
HepRef(PHisto1DVar) Histo1DVar(const char *,P_Points_Vector &,
              End_Point_Convention epc = RIGHT_OPEN );
HepRef(PProfileHisto) ProfileHisto1D (const char *a_title, Size n,
              double x1, double x2, End_Point_Convention  epc =
              RIGHT_OPEN );
HepRef(PProfileHistoVar) ProfileHisto1DVar(const char *,
              P_Points_Vector &, End_Point_Convention epc = RIGHT_OPEN );
```

[1] The SWIG interface generator provides a generic mechanism for integrating a wide variety of scripting languages, including Tcl, Perl, Python and even Java.

```
HepRef(PHisto2D) Histo2D (const char *, Size , double  , double ,
              Size , double  , double , End_Point_Convention  epc1 =
              RIGHT_OPEN , End_Point_Convention  epc2 = RIGHT_OPEN );
HepRef(PHisto2DVar) Histo2DVar( const char *, P_Points_Vector &,
              P_Points_Vector &, End_Point_Convention  epc1 =
              RIGHT_OPEN, End_Point_Convention   epc2 = RIGHT_OPEN );
```

6.2 Naming histograms

The HepODBMS package of LHC++ provides a logical naming scheme on top of the physical structure of the underlying database. Note that these two structures are independent - naming does not imply physical location and vice-versa. The naming package creates and maintains a naming tree similar to file-system trees, with directories and "final objects" (equivalent to files in a file-system).

The HepODBMS naming package assumes each user is given a personal naming tree which starts in a "home directory" like structure. Each user is assigned a home directory that maps directly to their user name under the common directory `/usr`. Thus, the home directory in the naming tree for a user named `dinofm` is `/usr/dinofm`.

An exampe of the use of the naming tree that will be familiar to HBOOK users would be to name HTL histograms using an identifier much like the HBOOK `ID` and store them in appropriate named directories in the database. This provides a naming scheme similiar to that used by HBOOK, with the exception that the naming is case sensitive.

Although users can use the HepODBMS classes directly to name HTL histograms, a simple `Factory` class which extends the basic `HFactory` with naming capabilities has also been provided. Using this extended factory class, users can choose where to store histograms (physical location) and name them according to their favourite scheme (logical location).

The following code fragment shows how HTL histograms can be named. Two histograms are created and stored in the same database and container as before. However, they are also given names in the `/usr/dinofm/Histograms/MC` directory - one using an HBOOK-like identifier and the other using a short text string.

```
#include "HTL/HNaming.h"
...
  HNamingFactory myFactory(HNamingFactory::Override);
  // Create histograms in DB "gepo", Container "sbaffini" and name
  // them in /usr/dinofm/Histograms/MC (dinofm is my user name)
  if (myFactory.init(this,"Histograms/MC","gepo","sbaffini")) {
    // Create two histograms. They're named 10 and "pt" in the name tree
    HepRef(PHisto1D) h1 = myFactory.Histo1D(10,"Histo-1",noBins,0.0,20.0);
    HepRef(PHisto1D) h2 = myFactory.Histo1D("pt","Histo-2",noBins,0.0,20.0);
    ...
    h1->fill(x, 1.);
    h2->fill(x, 2.);
```

Named histograms, such as those created above, can then be retrieved as follows:

```
/usr/dinofm/Histograms/MC/10
/usr/dinofm/Histograms/MC/pt
```

It should be remembered that - unlike HBOOK - the naming scheme is case sensitive. Histograms can be named with a character string such as "`pt`" or with an integer code like 10 (similar to HBOOK's histogram identifier `ID`).

If the histogram you create already exists, `HNamingFactory` can either override the previous histogram or refuse to create the new one, depending on the strategy specified when the factory was declared.

To instruct the factory to override existing histograms, use a construct like:

```
HNamingFactory myFactory(HNamingFactory::Override);
```

To avoid accidental overwriting of existing histograms, use something like:

```
HNamingFactory myFactory(HNamingFactory::Keep);
```

The Override directive can be useful for cases such as batch jobs or debugging: old histograms are simply replaced by new ones. The `Keep` mode allows one to protect against the accidental deletion of useful data. Other strategies, such as cycles or versions, are not presently implemented.

6.3 Retrieving named histograms

Once the histograms have been saved and named in an Objectivity data store, a mechanism for retrieving them is clearly required. The `HLocator` class allows the retrieval of a histogram by name, as follows:

```
#include "HTL/HNaming.h"
...
  // Locate Histograms pt and 10
  HLocator myLocator;
  HPrintert hp(cout);
  myLocator.init(this);
  HepRef(PHisto1D) h = myLocator.Histo1D("pt");
  if (h != 0)
    hp.print(*h);
  h = myLocator.Histo1D(10);
  if (h != 0)
    hp.print(*h);
```

Notice that the `HLocator` class provides methods to retrieve different kinds of histograms (so users should know what they are retrieving). Type checking is thus guaranteed and if the expected histogram type does not match the real one, an error message is printed and the return value is zero.

Chapter 7: Using interfaces

Interfaces are a powerful OO concept which allows designers to decouple what an object does from its actual implementation. This reduces the impact on "external" software using such object(s) and makes software reuse more realistic.

In the C++ world, interfaces are usually implemented by abstract classes with virtual methods. In this context the only drawback of interfaces might be a performance penalty due to the extra indirection required by virtual functions, but we should not neglect the usefulness of interfaces just because of that.

7.1 Interfaces in HTL

HTL defines and implements abstract interfaces for most of its functionalities. The only method which is not available on purpose at the interface level is the `fill()` method, which must be implemented as an non-virtual inline method to provide the highest performance (see Appendix C for details).

HTL interfaces are used by other packages such as fitting (HEPFitting) and visualisation (HEPInventor) that are thus independent from the actual HTL implementation of histograms. Interfaces are what allows external packages to deal with transient and persistent HTL histograms at the same time. Using such interfaces we might even implement an HTL-like package based on other technologies, for instance an HTL interface to HBOOK, although this is not envisaged at present.

7.1.1 The `I_Histo` interface

`I_Histo` is the high-level interface to histograms. The following is a list of supported methods.

`virtual const char* name()` Title attached to current histogram.

`virtual I_Bin& i_bin( I_Bin_Location &a_location )` In-range bin associated with location `a_location`.

`virtual I_Bin& i_extra_bin( I_Extra_Bin_Location &a_location )` Extra bin associated with extra location `a_location`.

`virtual Size bin_count()` Number of in-range bins.

`virtual Size extra_bin_count()` Number of extra bins.

`virtual Size dim()` Dimension of the histo, i.e., of the problem space.

`virtual I_Partition& i_partition( Index p = 0 )` Partition interface associated with this histo. For the first partition one has `a_dim_index = 0`.

`virtual I_Bin& i_bin( Index i )` Any bin (in-range or extra) with index "`i`" (note that this is a linear access).

The `I_Histo` interface can retrieve the number of bins that are in-range or out-range, for instance, overflow or underflow, as well as the number of partitions (i.e. the dimensionality of the histogram). It also allows you to access other interfaces such as `I_Partition` and `I_Bin`.

On the other hand, the `I_Histo` interface does not try to provide all information in a single interface: details about binning and bin content are delegated, respectively, to the `I_Partition` and `I_Bin` interfaces, as explained later.

7.1.2 The `I_Partition` interface

The `I_partition` interface deals with binning details, such as, which interval of the problem space a bin is mapped to, that is where does it start and end, and what are the limits of the range spanned by the partition. The following is a list of supported methods.

`End_Point_Convention end_point_convention()` End point convention for all bins; can be either `RIGHT_OPEN` or `LEFT_OPEN`.

`virtual double i_bin_width( Index i )` Width of in-range bin "`i`".

`virtual double i_lower_point()` Leftmost point of the partition.

`virtual double i_lower_point( Index i )` Leftmost point of bin indexed by "`i`".

`virtual double i_upper_point()` Rightmost point of the partition.

`virtual double i_upper_point( Index i )` R'ightmost point of bin indexed by "`i`".

7.1.3 The `I_Bin` interface

The `I_Bin` interface allows you to set or get the content of a bin, its count information, and its error. The available methods are listed below.

`virtual double value( Index i = 0)` Value associated with this bin.

`virtual double error( Index i = 0)` Error associated with this bin.

`virtual Size count()` Count associated with this bin. = Number of entries.

`virtual void set_value( double other, Index i = 0 )` Set the value associated with this bin to "`other`".

`virtual void set_error( double other, Index i = 0 )` Change/set the error of the bin to "`other`".

`virtual void set_count( Size other )` Change/set the count of the bin to "`other`".

`virtual double center( Index i = 0)` Absolute or relative center of this bin on axis "`i`".

`virtual int offset( Index i = 0)` Relative or absolute position for the center of the bin.

7.2 Example using Interface classes

The following example is derived from the HEPInventor visualisation package - only a subset of the complete code is used. The method receives an `I_Histo` reference which is then used, for instance, to find out whether this is a 1D or 2D histogram. Via the same `I_Histo` reference, the code retrieves an interface to the underlying partition, which gives information about the beginning and end of each bin. At the end, the `I_Bin` interface of each bin allows us to retrieve the content and the error associated to each bin.

```
HIData::HIData(I_Histo &histo) {
  int i,j,k=0;
  switch (histo.dim()) { // Is it 1D or 2D?
  case 1:
    // Copy the histogram content in a local data structure
    nptx       = histo.bin_count();
    // Use underlying histo.i_partition() interface
    Xval[0]    = histo.i_partition(0).i_lower_point(0);
    Yval[0]    = histo.i_bin(0).value();
    // Had to patch this... no asymmetric errors yet...
    EXval[0]   = (histo.i_partition(0).i_lower_point(1)-Xval[0])/2.;
    EYval[0]   = histo.i_bin(0).error();
    for ( i=1; i < nptx; i++) {
      Xval[i]  = histo.i_partition(0).i_lower_point(i);
      // Use I_Bin interface to retriev value/error
      Yval[i]  = histo.i_bin(i).value();
      EYval[i] = histo.i_bin(i).error();
    }
  }
}
```

As you can see there is no reference whatsoever to the real C++ type of the histogram, to its dimensionality, etc. The same code works with all HTL histograms!

Chapter 8: Constructors

8.1 Types of constructors

HTL provides more constructors than the current package and allows us to define and create new histograms out of transient or persistent ones in a nice and natural way.

Three kinds of constructor are defined for any histogram class.

The first kind of constructor is the classic one used for the booking. Arguments reflect the type of the histogram, for instance, 1D or 2D, fixed or variable bin size, etc. An example is the following:

```
Histo1D( const char *a_title, Size nBins, double lowX, double
          highX, End_Point_Convention epc = RIGHT_OPEN )
```

The second kind of constructor is a copy constructor. The first argument must be a histo of the same type. The optional second argument specifies whether we want to copy the structure only (the argument must equal 0) or the structure as well as the data also. (This is the default - argument non 0).

```
Histo1D( Like_Current &a_histo, int copy_data = 1 )
```

`Like_Current` is a typedef and designates the real type of the current histogram class with all its template arguments.

The third kind of constructor is a variant of a copy constructor. It makes it possible to create a new histogram out of a histo interface. It is particularly useful to create persistent histograms from transient ones. The optional second argument for data copying is also available.

```
Histo1D( const I_Histo &a_histo, int copy_data = 1 )
```

Chapter 9: Retrieving statistics and entries information

9.1 Shortcuts

1D histograms implement shortcut methods to retrieve mean and RMS of a histogram, as follows:

```
Histo1D *h1D = new Histo1D("Histo1: parabolic function",20,0.,20.);
...
cout << "Mean " << h1D->mean() << " RMS " << h1D->rms() << endl;
```

9.2 General method

HTL implements retrieval of statistics and entries information via separate helper classes. The helper class is called `HStat` and it allows us to decouple computation of statistics and entries information from the implementation of the histogram.

```
#include "HTL/Histograms.h"

Histo1D *h1D = new Histo1D("Histo1: parabolic function",20,0.,20.);

... // filling

// Now display some statistics:
//
long in_entries = HStat::in_range_entries_count(*histo);
long extra_entries = HStat::extra_entries_count(*histo);
cout << "Entries count: " << (in_entries + extra_entries)
 << " with IN_RANGE: " << in_entries
 << " EXTRA: " << extra_entries << endl;

double mean = HStat::mean( *histo );
cout << "MEAN (B.C.): " << mean << endl;
cout << "RMS  (B.C.): " << HStat::rms(*histo, mean)
 << endl << endl;
```

Appendix A: Supported compilers

The list of supported compilers on a given platform follows

Solaris CC 4.2

HP-UX aCC A.01.15

Linux egcs 1.1.1 with patch for persistent templates specialization

Digital Unix cxx V6.1-027

AIX xlC 3.1.?

Windows/NT VC++ 5.0 SP3

Appendix B: Statistics and error propagation

B.1 Histogram statistics

HTL implements histogram statistics as bin content statistics only.

```
Mean = SUM[ bin center * bin value ] / SUM[ bin value ]
RMS = SUM[ (bin center - mean)^2 * bin value ] / SUM[ bin value ]
```

Since the `Gravity_Bin` used in 1D HTL histograms keeps the weighted center of the bin, those quantities are equal to so-called “filling time statistics” computed by HBOOK and HistOOgrams.

B.2 Bin errors

Bin errors are always computed taking weights into account.

```
error = sqrt( SUM[ weight*weight ])
```

B.2.1 Bin error “propagation” for histogram operations

Depending on histogram operations, bin errors are combined in the error of the resulting bin. Formulas should be equivalent to the ones used in HBOOK.

B.2.2 Synopsis

`error_` error of the first bin, `other.error()` is the error of the second bin;

`value_` value of the first bin, `other.value()` is the error of the second bin.

```
 void add( const Like_Parent &other )
  error_ =  other.error()*other.error() + error_ ;

void sub( const Like_Parent &other )
  error_ = other.error() * other.error() + error_ ;

void mul( const Like_Parent &other )
  error_ = error_ * other.value()*other.value() +
           other.error()*other.error() * value_*value_ ;

void div( const Like_Parent &other )
  e = pow(other.error(),4) ;
  error_ = error_ * other.value()*other.value() +
           other.error()*other.error() * value_*value_ ;
  if( e != 0 )
    error_ = ( error_/e );

void binomial_div( const Like_Parent &other )
  e = other.error()*other.error() ;
  error_ = sqrt( error_ / e );
  error_ = error_ * (1-error_);
  error_ = error_ * error_;
  error_ = ( error_ / e );
```

Appendix C: Performance

C.1 Introduction

Most of the time the performance of an histogramming package is not critical. For example, in the typical case of long running batch jobs, the time spent in histogram operations is not that important. On the other hand there are applications, such as online monitoring, where excellent performance is fundamental.

The HistOOgram package, which HTL replaces, was not optimised for performance, but rather was designed for maximum flexibility. In addition, early benchmarks of this package were performed on a pre-release and should be considered unrepresentative.

Experience shows that there is always a tradeoff between higher performance and maximum flexibility. Fortunately, a reasonable compromise can usually be found, since performance is normally required only in well defined areas of the code.

A often-heard rule of thumb states that in most cases an application spends 80% of its time in 20% of the code. Hence an efficient approach is to estimate where the critical sections of the code lie, identify the most appropriate algorithms (which are more difficult to change than code), and finally measure the performance with a proper tool, such as a code profiler. Code portability and maintainability should not be abandoned in the pursuit of performance - all are important issues that need to be addressed when producing a package.

The procedure described above was that used in the case of HTL. In other words, the critical portions of code were first identified, which unsurprisingly turned out to be the filling methods, that can be called millions of times. A technique to speed up filling by using templated classes was then identified (see for instance the Blitz++ libraries for a discussion about templates and C++ performance). Finally, once the package was working, the performance was measured and tuned using a simple code profiler.

The results of a simple comparison with HBOOK are presented below. These are not intended as a complete test but rather as a benchmark reference. The source code used for the benchmark is available on request.

C.2 Comparison between HBOOK, HistOOgrams, and HTL

C.2.1 Description of the benchmark

A set of 10 histograms with fixed binning is filled with 50000 points each. The benchmark is focused on filling performance.

The I/O part is more difficult to compare directly and is not included in these comparisons. Objectivity/DB, used by persistent HTL, ensures that all data is written to disk as part of a transaction. In the case of HBOOK buffers are not flushed and so a comparison of the two is not meaningful. In the HBOOK case, timing is measured with the CERNLIB routine `TIMED`. The C++ benchmark reports real time as measured by RD45's `Timer` class. HTL histograms are based on gravity bins, so they harvest more information than their HBOOK counterpart, and thus use CPU.

C.2.2 Results of the benchmark

The target machine used for the test was a Sparc Ultra:

```
SunOS sunasd1 5.5.1 Generic_103640-12 sun4u sparc SUNW,Ultra-30
```

The compilers used were the following: `C++ V 4.1` for HistOO, `C++ V4.2` for HTL, and `f77 4.2` for HBOOK.

The results are summarised in the table below.

HBOOK	860 ms
Transient HTL	790 ms
HistOO	3600 ms
Persistent HTL	2700 ms
Persistent HTL(fast-filling)	1150 ms

The timing on the second line of the persistent HTL case is for a version optimised by using a fast filling method.

It should also be stressed that the HTL package allows you to create a persistent Histogram out of a transient one so that it is possible to work in a mixed mode: first histograms can be created and filled as transient, and only at the end saved as persistent. Thus one combines the advantages of a fastest possible filling with the those of profiting of object persistency.

C.3 Conclusion

HTL transient histograms are slightly faster than comparable HBOOK ones, even though they provide the user with more accurate information about bin centers.

HTL persistent histograms are somewhat slower than their transient counterparts. It would be possible, using more advanced Objectivity/DB techniques such as fast-filling methods and pinning in memory, to make them almost as fast as transient ones. However, a more practical approach in most cases is the use of transient histograms for repetitive operations such as filling, combined with persistent copies of the filled histogram for storage.

When raw performance is absolutely crucial, faster HTL histograms are also available. However, they do not provide filling time statistics.

Since HTL allows you to mix persistent and transient histograms, the optimal strategy when performance is an issue is to book and fill transient histogram and save them at the end of the run using the persistent histogram `copy` constructors (as explained in chapter 2).

Appendix D: Running the examples

D.1 List of available examples

A set of HTL examples can be found in the directory `file:/afs/cern.ch/sw/lhcxx/share/HTL/dev/HTL/examples`.

`Histo1D`	create persistent 1D histogram with fixed binning;
`Histo2D`	create persistent 2D histogram with fixed binning;
`HistoProfile`	create persistent 1D profile histogram with fixed binning;
`HistoTables`	produce ASCII tables to export HTL histograms to, e.g., PAW;
`Operations`	histogram cloning and histogram operations;
`THistos`	create transient 1D/2D histograms with fixed binning;
`Histo1DVar`	create persistent 1D histogram with variable binning;
`Histo2DVar`	create persistent 2D histogram with variable binning;
`HistoRandom`	use of CLHEP random generators;
`NameHistos`	naming persistent histograms;
`ReadHisto`	reading back histograms from Objectivity/DB datastore;
`TSliceProj`	slice/projections of a 2D histogram.

D.2 Setting up for using the examples

D.2.1 Defining your environment

When you are running in an environment created by the LHC++ setup procedure (see `http://wwwinfo.cern.ch/asd/lhc++/lhcppguide/SetUpUserEnvironment.html`) all environment variables needed by HTL will be already in place.

If you are not running in such an environment, you will have to set the HTL environment variables up yourself, since the GNUmakefiles for running the examples assume that the following environment variables are defined correctly:

- `LHCXXTOP`
- `PLATF`
- `HEP_ODBMS_DIR`
- `HISTOODIR`

If they are not defined run the following script which is likely to work at CERN (replace *pro* by *new* or *dev* to get the new or development versions as required):

```
source /afs/cern.ch/sw/lhcxx/share/HTL/pro/HTL.csh
```

The above syntax is for C-shell flavours, while on Bourne shell flavours you should use:

```
. /afs/cern.ch/sw/lhcxx/share/HTL/pro/HTL.ksh
```

For non-CERN installations you will have to replace the path:

```
/afs/cern.ch/sw/lhcxx
```

with the location of your LHC++ tree.
Finally, a similar script is provided on Windows/NT as a batch file:

```
Z:\P32\lhcxx\share\HTL\pro\HTL.bat
```

If you are using persistent HTL your Objectivity/DB environment should be properly defined as well.

D.2.2 Building the examples

Make a local copy of the examples directory and run `gmake`, as follows:

```
cp -r /afs/cern.ch/sw/lhcxx/HTL/dev/HTL/examples .cd examplesgmake
```

These commands will build all examples in one go. You can build examples separately by running `gmake` in each subdirectory.

D.2.3 Running the examples

HTL is provided as shared libraries on most platforms. This means HTL libraries and any other shared library used by HTL, such as HepODBMS and Objectivity/DB for persistent HTL, should be visible in your shared library path.
Provided that your environment is already configured for LHC++, no furher setup is required. If not, as a mininum your shared library path must include the directory where the HTL library (and also Objectivity/DB, for the persistent version) is located. The scripts mentioned in the previous sections will update your library path as well.
The examples executables are created in a `$(OS)` subdirectory in each example directory.

D.3 Structure of each example

Each example directory contains two files: a GNUmakefile to build the example and a source file containing the C++ code. The source file typically defines an "Application" class and a main program which instantiates that class and calls one or more methods.

```
class Histo_App :
...
int main( int argc, char **argv )
{
 Histo_App app;
 app.init();
 app.run();
 app.commit();
}
```

The "Application" class may inherit from `HepODBMS::HepDBApplication` when working with the persistent HTL examples. This allows the class to manage transactions and so on. Apart from transaction issues, most examples based on persistent HTL can be translated to transient HTL by changing all occurrences of `PHisto` to `Histo`

D.4 One dimensional histograms

The following example shows the creation, filling and line-printer output of a 1D histogram and some of its attributes. The histogram itself has weighted bins and an even partition. The histogram is persistent - that is, it is stored in the database and remains there after the job has completed.

D.4.1 Input program

```
/* create_Histo1D.cpp */
#include <iostream.h>
#include <iomanip.h>                 // Formatting output string.
#include "HTL/PHistograms.h"
class Histo_App : public HepDbApplication
{
public:
        Histo_App( const char *a_db_name = 0 )
        { db_name_ = ( a_db_name == 0 ) ? "Default_DB" : a_db_name; }
                // If no database name is provided then the default one is
                // "Default_DB".
public:
        void run();
                // Create a 1D histo named "Histo 1D" and fill it.
                // Print simple properties and global statistics.
public:
        void init();
private:
        ooHandle(ooDBObj) db_; // Handle on the database.
        const char *db_name_; // Name of the database.
};
//
// Implementation:
//
void Histo_App::init()
{
        Init();
        startUpdate();
        db_ = db(db_name_);
}
void Histo_App::run()
{
        const Size bin_count = 20;
        const double x_min = 0.0;
        const double x_max = 20.0;
        // Create a histo 1D using Weighted_Bin and Even_Partition:
        // (The type of the points is double.)
        //
        HepRef(PHisto1D) histo = new (db_) PHisto1D( "Histo_1D", bin_count, x_min, x_max );
        // Let's fill the histo with 50000 points:
        //
        long i;
        double x, w;
        for( i=0; i<50000; i++ ) {
                x = (i % 22) - 1;
                w = (x-9.5)*(x-9.5)+3;
                histo->fill(x,w);
        }
        // Let's print some properties of the new histo:
        //
        cout << "Histo name: " << histo->name() << endl
```

```
                << "Bin count : " << histo->bin_count()
                << " from " << histo->partition().lower_point()
                << " to " << histo->partition().upper_point() << endl << endl;
        // Now display some statistics:
        //
        long in_entries = HStat::in_range_entries_count(*histo);
        long extra_entries = HStat::extra_entries_count(*histo);
        cout << "Entries count: " << (in_entries + extra_entries)
                << " with IN_RANGE: " << in_entries
                << " EXTRA: " << extra_entries << endl;
        //double mean = HStat::mean( *histo );
        //cout << "MEAN (B.C.): " << mean << endl;
        //cout << "RMS  (B.C.): " << HStat::rms(*histo, mean)
        //        << endl << endl;
        cout << "MEAN (B.C.): " << histo->mean() << endl;
        cout << "RMS  (B.C.): " << histo->rms() << endl;
        // Print histo contents:
        //
        cout << ". UNDERFLOW: " << histo->extra_bin(H_UNDERFLOW).value();
        for( i=0; i<histo->bin_count(); i++ ) {
                cout << endl << ". " << setw(2) << i << ": "
                        << setw(9) << histo->i_bin(i).value() << " +/- "
                        << setw(4) << histo->i_bin(i).error();
        }
        cout << endl << ". OVERFLOW: " << histo->extra_bin(H_OVERFLOW).value();
        cout << endl <<  "TOTAL IN_RANGE: "
                << HInfo::in_range_value( *histo ) << endl << endl;
        // Print histo with existing HPrinter class from the library:
        //
        HPrinter hp( cout );
        hp.print( *histo );
}
int main( int argc, char **argv )
{
        Histo_App app;
        app.init();
        app.run();
        app.commit();
}
```

D.4.2 Output Generated

```
Histo name: Histo_1D
Bin count : 20 from 0 to 20
Entries count: 50000 with IN_RANGE: 45455 EXTRA: 4545
MEAN (B.C.): 9.49851
RMS  (B.C.): 7.58048
. UNDERFLOW: 257417
.  0:    211957 +/- 4445.79
.  1:    171043 +/- 3587.62
.  2:    134675 +/- 2824.8
.  3:    102853 +/- 2157.34
.  4:   75577.2 +/- 1585.23
.  5:   52847.2 +/- 1108.47
.  6:   34663.2 +/- 727.059
.  7:   21025.2 +/- 441.003
.  8:   11933.2 +/- 250.299
.  9:   7387.25 +/- 154.947
. 10:   7387.25 +/- 154.947
```

```
. 11:   11933.2 +/- 250.299
. 12:   21025.2 +/- 441.003
. 13:   34663.2 +/- 727.059
. 14:   52847.2 +/- 1108.47
. 15:     75544 +/- 1584.88
. 16:    102808 +/- 2156.86
. 17:    134616 +/- 2824.18
. 18:    170968 +/- 3586.83
. 19:    211864 +/- 4444.81
. OVERFLOW: 257304
TOTAL IN_RANGE: 1.64762e+06
TYPE       : Histo1D
TITLE      : Histo_1D
BIN COUNT  : 20
BIN WIDTH  : 1
                         7.39e+03                          2.12e+05 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     2.120e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+00   1     1.710e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   2     1.347e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     1.029e+05|XXXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     7.558e+04|XXXXXXXXXXXXXXXX
 5.000e+00   5     5.285e+04|XXXXXXXXXX
 6.000e+00   6     3.466e+04|XXXXXX
 7.000e+00   7     2.103e+04|XXX
 8.000e+00   8     1.193e+04|X
 9.000e+00   9     7.387e+03|
 1.000e+01   10    7.387e+03|
 1.100e+01   11    1.193e+04|X
 1.200e+01   12    2.103e+04|XXX
 1.300e+01   13    3.466e+04|XXXXXX
 1.400e+01   14    5.285e+04|XXXXXXXXXX
 1.500e+01   15    7.554e+04|XXXXXXXXXXXXXXX
 1.600e+01   16    1.028e+05|XXXXXXXXXXXXXXXXXXXXX
 1.700e+01   17    1.346e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    1.710e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   19    2.119e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                        X |
                          V
UNDERFLOW :  2.574e+05 OVERFLOW :  2.573e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  7.580e+00
```

D.5 One dimensional variable histograms

The following example shows the use of a variable partition, but is otherwise similar to the previous case.

D.5.1 Input program

```
/* create_Histo1DVar.cpp */
#include <iostream.h>
#include <iomanip.h>                // Formatting output string.
#include "HTL/PHistograms.h"        // Computing statistics.
class Histo_App : public HepDbApplication
{
public:
        Histo_App( const char *a_db_name = 0 )
        { db_name_ = ( a_db_name == 0 ) ? "Default_DB" : a_db_name; }
```

```
                // If no database name is provided then the default one is
                // "Default_DB".
public:
        void run();
                // Create a 1D histo named "Histo 1D" and fill it.
                // Print simple properties and global statistics.
public:
        void init();
private:
        ooHandle(ooDBObj) db_; // Handle on the database.
        const char *db_name_; // Name of the database.
};
//
// Implementation:
//
void Histo_App::init()
{
        Init();
        startUpdate();
        db_ = db(db_name_);
}
void Histo_App::run()
{
        const Size bin_count = 20;
        const double x_min = 0.0;
        const double x_max = 20.0;
        // Create a Histo1DVar using Weighted_Bin and Uneven_Partition:
        // (The type of the points is double.)
        //
        //
        //         1/ Declare and define the vector of points that compose the
        //         (uneven/variable) partition:
        //
        P_Points_Vector my_points(10);
        my_points[0] = 0.0; my_points[1] = 1.0;
        my_points[2] = 3.5; my_points[3] = 4.0;
        my_points[4] = 5.5; my_points[5] = 9.0;
        my_points[6] = 10.; my_points[7] = 15.;
        my_points[8] = 19.; my_points[9] = 20.;
        //         2/ Now declare and define the Histo1DVar:
        //
        HepRef(PHisto1DVar) histo = new (db_) PHisto1DVar( "Histo1DVar", my_points );
        // Let's fill the histo with 50000 points:
        //
        long i;
        double x, w;
        for( i=0; i<50000; i++ ) {
                x = (i % 22) - 1;
                w = (x-9.5)*(x-9.5)+3;
                histo->fill(x,w);
        }
        // Now print some properties of the new histo:
        //
        cout << "Histo name: " << histo->name() << endl
                << "Bin count : " << histo->bin_count()
                << " from " << histo->partition().lower_point()
                << " to " << histo->partition().upper_point() << endl << endl;
        // Now display some statistics:
        //
        long in_entries = HStat::in_range_entries_count(*histo);
        long extra_entries = HStat::extra_entries_count(*histo);
```

```
        cout << "Entries count: " << (in_entries + extra_entries)
                << " with IN_RANGE: " << in_entries
                << " EXTRA: " << extra_entries << endl;
        //double mean = HStat::mean( *histo );
        //cout << "MEAN (B.C.): " << mean << endl;
        //cout << "RMS  (B.C.): " << HStat::rms(*histo, mean)
        //        << endl << endl;
        cout << "MEAN (B.C.): " << histo->mean() << endl;
        cout << "RMS  (B.C.): " << histo->rms() << endl;
        // Print histo contents:
        //
        cout << ". UNDERFLOW: " << histo->extra_bin(H_UNDERFLOW).value();
        for( i=0; i<histo->bin_count(); i++ ) {
                cout << endl << ". " << setw(2) << i << ": "
                        << setw(9) << histo->i_bin(i).value() << " +/- "
                        << setw(4) << histo->i_bin(i).error();
        }
        cout << endl << ". OVERFLOW: " << histo->extra_bin(H_OVERFLOW).value();
        cout << endl <<  "TOTAL IN_RANGE: "
                << HInfo::in_range_value( *histo ) << endl << endl;
        // Print histo with existing HPrinter class from the library:
        //
        HPrinter hp( cout );
        hp.print( *histo );
}
int main( int argc, char **argv )
{
        Histo_App app;
        app.init();
        app.run();
        app.commit();
}
```

D.5.2 Output Generated

```
Histo name: Histo1DVar
Bin count : 9 from 0 to 20
Entries count: 50000 with IN_RANGE: 45455 EXTRA: 4545
MEAN (B.C.): 9.49851
RMS  (B.C.): 7.53709
. UNDERFLOW: 257417
.  0:    211957 +/- 4445.79
.  1:    408572 +/- 5050.21
.  2:         0 +/-    0
.  3:    128424 +/- 1934.33
.  4:   67621.8 +/- 886.424
.  5:   7387.25 +/- 154.947
.  6:    127856 +/- 1427.74
.  7:    483936 +/- 5292
.  8:    211864 +/- 4444.81
. OVERFLOW: 257304
TOTAL IN_RANGE: 1.64762e+06
TYPE       : Histo1D
TITLE      : Histo1DVar
BIN COUNT  : 9
BIN WIDTH  : 1
                          0.00e+00                          4.84e+05 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     2.120e+05|XXXXXXXXXXXXXXXXXXXXX
```

```
 1.000e+00   1     4.086e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.500e+00   2     0.000e+00|
 4.000e+00   3     1.284e+05|XXXXXXXXXXXXX
 5.500e+00   4     6.762e+04|XXXXXXX
 9.000e+00   5     7.387e+03|
 1.000e+01   6     1.279e+05|XXXXXXXXXXXXX
 1.500e+01   7     4.839e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   8     2.119e+05|XXXXXXXXXXXXXXXXXXXX
                        X |
                          V
UNDERFLOW :  2.574e+05 OVERFLOW :  2.573e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  7.537e+00
```

D.6 Two dimensional histograms

The following example shows the use of a 2D histogram with fixed partitions.

D.6.1 Input program

```
/* create_Histo2D.cpp */
#include <iostream.h>
#include "HTL/PHistograms.h"         // Persistent histograms.
class Histo_App : public HepDbApplication
{
public:
        Histo_App( const char *a_db_name = 0 )
        { db_name_ = ( a_db_name == 0 ) ? "Default_DB" : a_db_name; }
                // If no database name is provided then the default one is
                // "Default_DB".
public:
        void run();
                // Create a 1D histo named "Histo 1D" and fill it.
                // Print simple properties and global statistics.
public:
        void init();
private:
        ooHandle(ooDBObj) db_; // Handle on the database.
        const char *db_name_; // Name of the database.
};
//
// Implementation:
//
void Histo_App::init()
{
        Init();
        startUpdate();
        db_ = db(db_name_);
}
void Histo_App::run()
{
        const int n_bin = 10;
        // Create a 2D histo using Weighted_Bin and Even_Partition:
        // (The type of the points is double.)
        //
        HepRef(PHisto2D) histo = new (db_) PHisto2D( "Histo_2D", n_bin, 5., 15., n_bin, 5., 15. );
        // Let's fill the histo with 50000 points:
        //
```

```
        double x, w = 0.5;
        for( long i=0; i<50000; i++ ) {
                x = -i*sin(float(i));
                histo->fill(x,x,w);
                histo->fill(20.-x,x,w);
        }
        // Let's print some properties of the new histo:
        //
        cout << "Histo name: " << histo->name() << endl
                << "X: Bin count : " << histo->partition_X().bin_count()
                << " from " << histo->partition_X().lower_point()
                << " to "   << histo->partition_X().upper_point() << endl;
        cout << "Y: Bin count : " << histo->partition_Y().bin_count()
                << " from " << histo->partition_Y().lower_point()
                << " to "   << histo->partition_Y().upper_point() << endl;
        // Now some statistics:
        //
        long in_entries = HStat::in_range_entries_count( *histo );
        long extra_entries = HStat::extra_entries_count( *histo );
        cout << "Entries count: " << (in_entries + extra_entries)
                << " with IN_RANGE: " << in_entries
                << " EXTRA: " << extra_entries << endl;
        cout << "In Range values: " << H_Bin_Helper::in_range_value( *histo )
                << endl;
        // Print histo contents:
        //
#define EH(h,i,j)         h->extra_bin(i,j).value()
        cout << endl;
        cout << "(U,O): " << EH( histo, H_UNDERFLOW, H_OVERFLOW )
                << " (I,O): " << EH( histo, H_IN_RANGE, H_OVERFLOW )
                << " (O,O): " << EH( histo, H_OVERFLOW, H_OVERFLOW ) << endl;
        cout << "(U,I): " << EH( histo, H_UNDERFLOW, H_IN_RANGE )
                << " (I,I): " << "XXXXXX"
                << " (O,I): " << EH( histo, H_OVERFLOW, H_IN_RANGE ) <<endl;
        cout << "(U,U): " << EH( histo, H_UNDERFLOW, H_UNDERFLOW )
                << " (I,U): " << EH( histo, H_IN_RANGE, H_UNDERFLOW )
                << " (O,U): " << EH( histo, H_OVERFLOW, H_UNDERFLOW ) << endl;
        cout << endl;
        for( long y=0; y<histo->partition_Y().bin_count(); y++ )
        for( long x=0; x<histo->partition_X().bin_count(); x++ ) {
                if( histo->bin(x,y).value() != 0.0 ) {
                        cout << "bin(" << x << "," << y << "): "
                                << histo->bin(x,y).value() << " -/+ "
                                << histo->bin(x,y).error() << endl;
                }
        }
        cout << endl;
        // Print using HPrinter:
        //
        HPrinter hp( cout );
        hp.print( *histo );
}
int main( int argc, char **argv )
{
        Histo_App app;
        app.init();
        app.run();
        app.commit();
}
```

D.6.2 Output Generated

```
Histo name: Histo_2D
X: Bin count : 10 from 5 to 15
Y: Bin count : 10 from 5 to 15
Entries count: 100000 with IN_RANGE: 58 EXTRA: 99942
In Range values: 29
(U,O): 12475 (I,O): 0 (O,O): 12475
(U,I): 0 (I,I): XXXXXX (O,I): 0
(U,U): 12510.5 (I,U): 0 (O,U): 12510.5
bin(0,0): 1 -/+ 0.707107
bin(9,0): 1 -/+ 0.707107
bin(1,1): 2 -/+ 1
bin(8,1): 2 -/+ 1
bin(2,2): 2 -/+ 1
bin(7,2): 2 -/+ 1
bin(3,3): 0.5 -/+ 0.5
bin(6,3): 0.5 -/+ 0.5
bin(4,4): 1 -/+ 0.707107
bin(5,4): 1 -/+ 0.707107
bin(4,5): 2 -/+ 1
bin(5,5): 2 -/+ 1
bin(3,6): 1 -/+ 0.707107
bin(6,6): 1 -/+ 0.707107
bin(2,7): 2 -/+ 1
bin(7,7): 2 -/+ 1
bin(1,8): 1.5 -/+ 0.866025
bin(8,8): 1.5 -/+ 0.866025
bin(0,9): 1.5 -/+ 0.866025
bin(9,9): 1.5 -/+ 0.866025
TYPE     : Histo2D
TITLE    : Histo_2D
X:  BINS :    10  WIDTH  : 1.000e+00  MIN    : 5.000e+00  MAX    : 1.500e+01
Y:  BINS :    10  WIDTH  : 1.000e+00  MIN    : 5.000e+00  MAX    : 1.500e+01
         0
         0123456789
        ************
       *            *
   0   *  7........7  *
   1   *  .F......F.  *
   2   *  ..F....F..  *
   3   *  ...3..3...  *
   4   *  ....77....  *
   5   *  ....FF....  *
   6   *  ...7..7...  *
   7   *  ..F....F..  *
   8   *  .B......B.  *
   9   *  B........B  *
        *            *
         ************
ENTRIES  :     100000   Z MIN    :          0
TOTAL C. :         29   Z STEP   :      0.133
Z SCALE  : .+23456789ABCDE
      1.25e+04 |          0 |   1.25e+04
    -----------|------------|-----------
             0 |         29 |          0
    -----------|------------|-----------
      1.25e+04 |          0 |   1.25e+04
```

D.7 Two dimensional variable histograms

The following example shows the use of a 2D histogram with variable partitions.

D.7.1 Input program

```
/* create_Histo2DVar.cpp */
#include <iostream.h>
#include "HTL/PHistograms.h" // Persistent histograms.
class Histo_App : public HepDbApplication
{
public:
        Histo_App( const char *a_db_name = 0 )
        { db_name_ = ( a_db_name == 0 ) ? "Default_DB" : a_db_name; }
                // If no database name is provided then the default one is
                // "Default_DB".
public:
        void run();
                // Create a 2D histogram with variable binning and fill it.
                // Print simple properties and global statistics.
public:
        void init();
private:
        ooHandle(ooDBObj) db_; // Handle on the database.
        const char *db_name_; // Name of the database.
};
//
// Implementation:
//
void Histo_App::init()
{
        Init();
        startUpdate();
        db_ = db(db_name_);
}
void Histo_App::run()
{
        //
        // Create a 2D histo using Weighted_Bin and Uneven_Partition:
        // (The type of the points is double.)
        //
        //
        //         1/ Declare and define the vectors of points that compose each
        //         of the (uneven/variable) partitions:
        //
        P_Points_Vector my_points_on_X(10);
        my_points_on_X[0] = 0.0; my_points_on_X[1] = 1.0;
        my_points_on_X[2] = 3.5; my_points_on_X[3] = 4.0;
        my_points_on_X[4] = 5.5; my_points_on_X[5] = 9.0;
        my_points_on_X[6] = 10.; my_points_on_X[7] = 15.;
        my_points_on_X[8] = 19.; my_points_on_X[9] = 20.;
        P_Points_Vector my_points_on_Y(10);
        my_points_on_Y[0] = 0.0; my_points_on_Y[1] = 1.0;
        my_points_on_Y[2] = 2.5; my_points_on_Y[3] = 4.0;
        my_points_on_Y[4] = 6.0; my_points_on_Y[5] = 9.0;
        my_points_on_Y[6] = 12.; my_points_on_Y[7] = 14.;
        my_points_on_Y[8] = 18.; my_points_on_Y[9] = 20.;
        //         2/ Now declare and define the Histo2DVar:
        //
```

```
        HepRef(PHisto2DVar) histo = new (db_) PHisto2DVar("Histo2DVar", my_points_on_X, my_points_on_Y );
        // Let's fill the histo with 50000 points:
        //
        double x, w = 0.5;
        for( long i=0; i<50000; i++ ) {
                x = -i*sin(float(i));
                histo->fill(x,x,w);
                histo->fill(20.-x,x,w);
        }
        // Let's print some properties of the new histo:
        //
        cout << "Histo name: " << histo->name() << endl
                << "X: Bin count : " << histo->partition_X().bin_count()
                << " from " << histo->partition_X().lower_point()
                << " to "   << histo->partition_X().upper_point() << endl;
        cout << "Y: Bin count : " << histo->partition_Y().bin_count()
                << " from " << histo->partition_Y().lower_point()
                << " to "   << histo->partition_Y().upper_point() << endl;
        // Now some statistics:
        //
        long in_entries = HStat::in_range_entries_count( *histo );
        long extra_entries = HStat::extra_entries_count( *histo );
        cout << "Entries count: " << (in_entries + extra_entries)
                << " with IN_RANGE: " << in_entries
                << " EXTRA: " << extra_entries << endl;
        cout << "In Range values: " << H_Bin_Helper::in_range_value( *histo )
                << endl;
        // Print histo contents:
        //
#define EH(h,i,j)         h->extra_bin(i,j).value()
        cout << endl;
        cout << "(U,O): " << EH( histo, H_UNDERFLOW, H_OVERFLOW )
                << " (I,O): " << EH( histo, H_IN_RANGE, H_OVERFLOW )
                << " (O,O): " << EH( histo, H_OVERFLOW, H_OVERFLOW ) << endl;
        cout << "(U,I): " << EH( histo, H_UNDERFLOW, H_IN_RANGE )
                << " (I,I): " << "XXXXXX"
                << " (O,I): " << EH( histo, H_OVERFLOW, H_IN_RANGE ) <<endl;
        cout << "(U,U): " << EH( histo, H_UNDERFLOW, H_UNDERFLOW )
                << " (I,U): " << EH( histo, H_IN_RANGE, H_UNDERFLOW )
                << " (O,U): " << EH( histo, H_OVERFLOW, H_UNDERFLOW ) << endl;
        cout << endl;
        for( long y=0; y<histo->partition_Y().bin_count(); y++ )
        for( long x=0; x<histo->partition_X().bin_count(); x++ ) {
                if( histo->bin(x,y).value() != 0.0 ) {
                        cout << "bin(" << x << "," << y << "): "
                                << histo->bin(x,y).value() << " -/+ "
                                << histo->bin(x,y).error() << endl;
                }
        }
        cout << endl;
        // Print using HPrinter:
        //
        HPrinter hp( cout );
        hp.print( *histo );
}
int main( int argc, char **argv )
{
        Histo_App app;
        app.init();
        app.run();
        app.commit();
```

```
}
```

D.7.2 Output Generated

```
Histo name: Histo2DVar
X: Bin count : 9 from 0 to 20
Y: Bin count : 9 from 0 to 20
Entries count: 100000 with IN_RANGE: 123 EXTRA: 99877
In Range values: 61.5
(U,O): 12469 (I,O): 0 (O,O): 12469
(U,I): 0 (I,I): XXXXXX (O,I): 0.5
(U,U): 12500 (I,U): 0 (O,U): 12500
bin(0,0): 3.5 -/+ 1.32288
bin(8,0): 3 -/+ 1.22474
bin(1,1): 2.5 -/+ 1.11803
bin(7,1): 2.5 -/+ 1.11803
bin(1,2): 2 -/+ 1
bin(2,2): 0.5 -/+ 0.5
bin(7,2): 2.5 -/+ 1.11803
bin(3,3): 2.5 -/+ 1.11803
bin(4,3): 0.5 -/+ 0.5
bin(6,3): 1 -/+ 0.707107
bin(7,3): 2 -/+ 1
bin(4,4): 4.5 -/+ 1.5
bin(6,4): 4.5 -/+ 1.5
bin(4,5): 1 -/+ 0.707107
bin(5,5): 3 -/+ 1.22474
bin(6,5): 4 -/+ 1.41421
bin(4,6): 3.5 -/+ 1.32288
bin(6,6): 3.5 -/+ 1.32288
bin(1,7): 0.5 -/+ 0.5
bin(2,7): 1 -/+ 0.707107
bin(3,7): 3 -/+ 1.22474
bin(6,7): 1.5 -/+ 0.866025
bin(7,7): 3 -/+ 1.22474
bin(0,8): 2.5 -/+ 1.11803
bin(1,8): 0.5 -/+ 0.5
bin(7,8): 0.5 -/+ 0.5
bin(8,8): 2.5 -/+ 1.11803
TYPE     : Histo2D
TITLE    : Histo2DVar
X:  BINS :     9  WIDTH  : 1.000e+00  MIN    : 0.000e+00  MAX    : 2.000e+01
Y:  BINS :     9  WIDTH  : 1.000e+00  MIN    : 0.000e+00  MAX    : 2.000e+01
         012345678
        ***********
       *           *
   0   *  B.......9  *
   1   *  .8.....8.  *
   2   *  .6+....8.  *
   3   *  ...8+.36.  *
   4   *  ....F.F..  *
   5   *  ....39D..  *
   6   *  ....B.B..  *
   7   *  .+39..49.  *
   8   *  8+.....+8  *
       *           *
        ***********
ENTRIES  :     100000   Z MIN    :          0
TOTAL C. :       61.5   Z STEP   :        0.3
```

```
Z SCALE  : .+23456789ABCDE
      1.25e+04 |          0 |   1.25e+04
    -----------|------------|-----------
             0 |       61.5 |        0.5
    -----------|------------|-----------
      1.25e+04 |          0 |   1.25e+04
```

D.8 Profile histograms

The following example shows the use of profile historams.

D.8.1 Input program

```
/* create_profile.cpp */
#include <stdlib.h>
#include <iostream.h>
#include <iomanip.h>                 // Formatting output string.
#include "CLHEP/Random/Randomize.h"
#include "HTL/PHistograms.h" // Persistent histograms.
class Histo_App : public HepDbApplication
{
public:
        Histo_App( const char *a_db_name = 0 )
        { db_name_ = ( a_db_name == 0 ) ? "Default_DB" : a_db_name; }
                // If no database name is provided then the default one is
                // "Default_DB".
public:
        void run();
                // Create a profile histo and fill it.
                // Print simple properties and global statistics.
public:
        void init();
private:
        ooHandle(ooDBObj) db_; // Handle on the database.
        const char *db_name_; // Name of the database.
};
//
// Implementation:
//
void Histo_App::init()
{
        Init();
        startUpdate();
        db_ = db(db_name_);
}
void Histo_App::run()
{
  // Create a profile histogram
  HepRef(PProfileHisto) histo = new (db_) PProfileHisto ("Profile", 20, 0., 1.);
  // Let's fill the histo with 50000 points:
  //
  long i;
  double x;
  for( i=0; i<50000; i++ ) {
    double x  = RandFlat::shoot(1.);       // fnum  ]0,1.[
    histo->fill(x,x,1.);
  }
  // Let's print some properties of the new histo:
```

```
  //
  cout << "Histo name: " << histo->name() << endl
       << "Bin count : " << histo->bin_count()
       << " from " << histo->partition().lower_point()
       << " to " << histo->partition().upper_point() << endl << endl;
        // Now display some statistics:
        //
  long in_entries = HStat::in_range_entries_count(*histo);
  long extra_entries = HStat::extra_entries_count(*histo);
  cout << "Entries count: " << (in_entries + extra_entries)
       << " with IN_RANGE: " << in_entries
       << " EXTRA: " << extra_entries << endl;
        //double mean = HStat::mean( *histo );
        //cout << "MEAN (B.C.): " << mean << endl;
        //cout << "RMS  (B.C.): " << HStat::rms(*histo, mean)
        //        << endl << endl;
  cout << "MEAN (B.C.): " << histo->mean() << endl;
  cout << "RMS  (B.C.): " << histo->rms() << endl;
        // Print histo contents:
        //
  cout << ". UNDERFLOW: " << histo->extra_bin(H_UNDERFLOW).value();
  for( i=0; i<histo->bin_count(); i++ ) {
    cout << endl << ". " << setw(2) << i << ": "
         << setw(9) << histo->i_bin(i).value() << " Error "
         << setw(4) << histo->i_bin(i).error() << " Spread Error "
         << setw(4) << histo->i_bin(i).error(1);
  }
  cout << endl << ". OVERFLOW: " << histo->extra_bin(H_OVERFLOW).value();
  cout << endl <<  "TOTAL IN_RANGE: "
       << HInfo::in_range_value( *histo ) << endl << endl;
}
int main( int argc, char **argv )
{
  Histo_App app;
  app.init();
  app.run();
  app.commit();
}
```

D.8.2 Output Generated

```
Histo name: Profile
Bin count : 20 from 0 to 1
Entries count: 50000 with IN_RANGE: 50000 EXTRA: 0
MEAN (B.C.): 0.666225
RMS  (B.C.): 0.235584
. UNDERFLOW: 0
.  0: 0.0246313 Error 0.000303054 Spread Error 0.0145213
.  1: 0.0752091 Error 0.000286633 Spread Error 0.0142972
.  2:   0.12533 Error 0.000282329 Spread Error 0.01436
.  3:   0.17502 Error 0.000290731 Spread Error 0.014452
.  4:  0.225639 Error 0.000283451 Spread Error 0.0142967
.  5:   0.27498 Error 0.000289269 Spread Error 0.0143677
.  6:  0.324899 Error 0.000285322 Spread Error 0.0143287
.  7:  0.374806 Error 0.000288683 Spread Error 0.0146092
.  8:  0.425111 Error 0.000286534 Spread Error 0.0143953
.  9:  0.474468 Error 0.000288799 Spread Error 0.0143995
. 10:  0.525262 Error 0.000283128 Spread Error 0.0142945
. 11:  0.574666 Error 0.000292235 Spread Error 0.0145297
```

```
. 12:  0.625178 Error 0.000283092 Spread Error 0.0144016
. 13:  0.675176 Error 0.000286276 Spread Error 0.0145325
. 14:  0.724437 Error 0.000294247 Spread Error 0.0144061
. 15:  0.774756 Error 0.000290449 Spread Error 0.0144584
. 16:  0.824792 Error 0.000286015 Spread Error 0.0143863
. 17:  0.874996 Error 0.00028511 Spread Error 0.0142641
. 18:  0.924813 Error 0.000288151 Spread Error 0.014318
. 19:  0.975295 Error 0.000288324 Spread Error 0.0143902
. OVERFLOW: 0
TOTAL IN_RANGE: 9.99947
```

D.9 Filling histograms with random numbers

The following example shows cloning of and operations on histograms.

D.9.1 Input program

```
/* histo_random.cpp */
#include <iostream.h>
#include "CLHEP/Random/Randomize.h"
#include "HTL/PHistograms.h" // Persistent histograms.
typedef PHisto1D PHisto;
class Histo_App : public HepDbApplication {
public:
  Histo_App( const char *a_db_name = 0 )
  { db_name_ = ( a_db_name == 0 ) ? "Default_DB" : a_db_name; }
  // If no database name is provided then the default one is
  // "Default_DB".
public:
        void run();
        void init();
private:
        ooHandle(ooDBObj) db_; // Handle on the database.
        const char *db_name_; // Name of the database.
};
//
// Implementation:
//
void Histo_App::init()
{
        Init();
        startUpdate();
        db_ = db(db_name_);
}
void Histo_App::run()
{
  const int Nbin = 10;
  Init();        // initialise the db session
  startUpdate(); // start an update transaction
  HepRef(PHisto) histo = new (db_) PHisto( "Histo_1D", 20., 0.0, 20.0 );
  HepRef(PHisto) xhist = new (db_) PHisto(" Random Numbers ",40,0.0,20.0);
  // Clone histogram
  HepRef(PHisto) x2hist = xhist->clone();
  x2hist->set_name("Two Gaussians");
  /* -----Start of histogram manipulation--------------------------------- */
  // Fill histograms with random number
  for(int i = 1;i <= 100000;i++)
    {
```

```
      double x  = RandFlat::shoot(20.);       // fnum  ]0,20[
      double n1 = RandGauss::shoot(5.,1.6);   // (mean=5, stDev=1.6)
      double n2 = RandGauss::shoot(15.2,0.7); // (mean=15.2, stDev=.7)
      xhist->fill(x,1.);
      x2hist->fill(n1,1.);
      x2hist->fill(n2,0.5);
    }
  cout << "Graphical Histogram Printout" << endl;

  HPrinter p(cout);

  p.print(*xhist); cout << endl;
  p.print(*x2hist);
  // Add x2hist to xhist, then divide by 2

  x2hist->add(*xhist);
  x2hist->div(2.);
  cout << endl
       << "Graphical Histogram Printout of xhist2 after operations"
       << endl;
  x2hist->set_name("Two Gaussians + Random divided by 2");
  p.print(*x2hist);


}
int main( int argc, char **argv )
{
        Histo_App app;
        app.init();
        app.run();
        app.commit();
}
```

D.9.2 Output Generated

```
Graphical Histogram Printout
TYPE       : Histo1D
TITLE      :  Random Numbers
BIN COUNT  : 40
BIN WIDTH  : 0.5
                          2.37e+03                          2.61e+03 Y
X POSITION   BIN       VALUE|---------------------------------------------------->
 0.000e+00   0     2.442e+03|XXXXXXXXXXXXXXX
 5.000e-01   1     2.452e+03|XXXXXXXXXXXXXXXX
 1.000e+00   2     2.443e+03|XXXXXXXXXXXXXXX
 1.500e+00   3     2.531e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   4     2.600e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.500e+00   5     2.535e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   6     2.503e+03|XXXXXXXXXXXXXXXXXXXXXXXXX
 3.500e+00   7     2.453e+03|XXXXXXXXXXXXXXXX
 4.000e+00   8     2.605e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 4.500e+00   9     2.608e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 5.000e+00   10    2.535e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 5.500e+00   11    2.529e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 6.000e+00   12    2.435e+03|XXXXXXXXXXXXX
 6.500e+00   13    2.513e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 7.000e+00   14    2.440e+03|XXXXXXXXXXXXXX
 7.500e+00   15    2.432e+03|XXXXXXXXXXXX
 8.000e+00   16    2.561e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
 8.500e+00   17    2.497e+03|XXXXXXXXXXXXXXXXXXXXXXXXX
 9.000e+00   18    2.602e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 9.500e+00   19    2.533e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+01   20    2.451e+03|XXXXXXXXXXXXXXXX
 1.050e+01   21    2.582e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.100e+01   22    2.533e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.150e+01   23    2.427e+03|XXXXXXXXXX
 1.200e+01   24    2.480e+03|XXXXXXXXXXXXXXXXXXXX
 1.250e+01   25    2.495e+03|XXXXXXXXXXXXXXXXXXXXXX
 1.300e+01   26    2.480e+03|XXXXXXXXXXXXXXXXXXXX
 1.350e+01   27    2.413e+03|XXXXXXXX
 1.400e+01   28    2.460e+03|XXXXXXXXXXXXXXXXX
 1.450e+01   29    2.475e+03|XXXXXXXXXXXXXXXXXXX
 1.500e+01   30    2.490e+03|XXXXXXXXXXXXXXXXXXXXX
 1.550e+01   31    2.480e+03|XXXXXXXXXXXXXXXXXXX
 1.600e+01   32    2.481e+03|XXXXXXXXXXXXXXXXXXX
 1.650e+01   33    2.510e+03|XXXXXXXXXXXXXXXXXXXXXXXX
 1.700e+01   34    2.372e+03|
 1.750e+01   35    2.555e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   36    2.484e+03|XXXXXXXXXXXXXXXXXXX
 1.850e+01   37    2.515e+03|XXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   38    2.502e+03|XXXXXXXXXXXXXXXXXXXXXXX
 1.950e+01   39    2.566e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                       X |
                         V
UNDERFLOW :  0.000e+00 OVERFLOW :  0.000e+00
 IN RANGE :     100000 EXTRA    :          0
 MEAN B.C.:  9.985e+00 RMS B.C. :  5.772e+00
TYPE       : Histo1D
TITLE      : Two Gaussians
BIN COUNT  : 40
BIN WIDTH  : 5.000e-01
                          0.00e+00                          1.39e+04 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     1.750e+02|
 5.000e-01   1     3.800e+02|X
 1.000e+00   2     8.020e+02|XX
 1.500e+00   3     1.586e+03|XXXXX
 2.000e+00   4     2.834e+03|XXXXXXXXXX
 2.500e+00   5     4.641e+03|XXXXXXXXXXXXXXXX
 3.000e+00   6     6.915e+03|XXXXXXXXXXXXXXXXXXXXXXXX
 3.500e+00   7     9.047e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 4.000e+00   8     1.104e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 4.500e+00   9     1.227e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 5.000e+00   10    1.237e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 5.500e+00   11    1.127e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 6.000e+00   12    9.228e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 6.500e+00   13    6.804e+03|XXXXXXXXXXXXXXXXXXXXXX
 7.000e+00   14    4.687e+03|XXXXXXXXXXXXXXXX
 7.500e+00   15    2.864e+03|XXXXXXXXXX
 8.000e+00   16    1.584e+03|XXXXX
 8.500e+00   17    8.010e+02|XX
 9.000e+00   18    3.860e+02|X
 9.500e+00   19    1.460e+02|
 1.000e+01   20    6.500e+01|
 1.050e+01   21    1.700e+01|
 1.100e+01   22    7.000e+00|
 1.150e+01   23    0.000e+00|
 1.200e+01   24    5.000e-01|
 1.250e+01   25    4.400e+01|
 1.300e+01   26    3.340e+02|X
```

```
 1.350e+01   27    1.770e+03|XXXXXX
 1.400e+01   28    5.772e+03|XXXXXXXXXXXXXXXXXXXX
 1.450e+01   29    1.153e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.500e+01   30    1.391e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.550e+01   31    1.033e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.600e+01   32    4.760e+03|XXXXXXXXXXXXXXXX
 1.650e+01   33    1.296e+03|XXXX
 1.700e+01   34    2.370e+02|
 1.750e+01   35    2.350e+01|
 1.800e+01   36    1.500e+00|
 1.850e+01   37    0.000e+00|
 1.900e+01   38    0.000e+00|
 1.950e+01   39    0.000e+00|
                        X |
                          V
UNDERFLOW :  8.600e+01 OVERFLOW :  0.000e+00
 IN RANGE :     199914 EXTRA    :         86
 MEAN B.C.:  8.407e+00 RMS B.C. :  4.991e+00
Graphical Histogram Printout of xhist2 after operations
TYPE       : Histo1D
TITLE      : Two Gaussians + Random divided by 2
BIN COUNT  : 40
BIN WIDTH  : 5.000e-01
                          1.21e+03                        8.20e+03 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     1.308e+03|
 5.000e-01   1     1.416e+03|X
 1.000e+00   2     1.622e+03|XX
 1.500e+00   3     2.058e+03|XXXXXX
 2.000e+00   4     2.717e+03|XXXXXXXXXX
 2.500e+00   5     3.588e+03|XXXXXXXXXXXXXXXX
 3.000e+00   6     4.709e+03|XXXXXXXXXXXXXXXXXXXXXXXX
 3.500e+00   7     5.750e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 4.000e+00   8     6.821e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 4.500e+00   9     7.438e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 5.000e+00   10    7.452e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 5.500e+00   11    6.900e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 6.000e+00   12    5.832e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 6.500e+00   13    4.658e+03|XXXXXXXXXXXXXXXXXXXXXXX
 7.000e+00   14    3.564e+03|XXXXXXXXXXXXXXXX
 7.500e+00   15    2.648e+03|XXXXXXXXXX
 8.000e+00   16    2.072e+03|XXXXXX
 8.500e+00   17    1.649e+03|XXX
 9.000e+00   18    1.494e+03|XX
 9.500e+00   19    1.340e+03|
 1.000e+01   20    1.258e+03|
 1.050e+01   21    1.300e+03|
 1.100e+01   22    1.270e+03|
 1.150e+01   23    1.214e+03|
 1.200e+01   24    1.240e+03|
 1.250e+01   25    1.270e+03|
 1.300e+01   26    1.407e+03|X
 1.350e+01   27    2.092e+03|XXXXXX
 1.400e+01   28    4.116e+03|XXXXXXXXXXXXXXXXXXXX
 1.450e+01   29    7.000e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.500e+01   30    8.200e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.550e+01   31    6.404e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.600e+01   32    3.620e+03|XXXXXXXXXXXXXXXX
 1.650e+01   33    1.903e+03|XXXXX
 1.700e+01   34    1.304e+03|
 1.750e+01   35    1.289e+03|
```

```
 1.800e+01   36    1.243e+03|
 1.850e+01   37    1.258e+03|
 1.900e+01   38    1.251e+03|
 1.950e+01   39    1.283e+03|
                         X |
                           V
UNDERFLOW :  4.300e+01 OVERFLOW :  0.000e+00
 IN RANGE :     299914 EXTRA    :         86
 MEAN B.C.:  9.039e+00 RMS B.C. :  5.373e+00
```

D.10 Saving histogram as text table

The following example shows how a histogram can be saved in a simple text file that can subsequently be read by PAW, Excel or other common tool. An example of a suitable PAW macro is also provided.

D.10.1 Input program

```
/* histo_tables.cpp */
#include <iostream.h>
#include <iomanip.h>                  // Formatting output string.
#include "HTL/Histograms.h" // Transient histograms.
class Histo_App
{
public:
  Histo_App() {}
public:
  void run();
  // Create one 1D transient histograms and save it a text table
private:
  void create1D ();
};
//
// Implementation:
//
void Histo_App::run() {
  create1D();
}
void Histo_App::create1D() {
  // Create a histo 1D
  Histo1D *histo = new Histo1D("Histo_1D", 20, 0.0, 20.0 );
  // Let's fill the histo with 50000 points:
  //
  long i;
  double x, w;
  for( i=0; i<50000; i++ ) {
    x = (i % 22) - 1;
    w = (x-9.5)*(x-9.5)/100;
    histo->fill(x,w);
  }
  // Save the histogram in a ASCII table
  HistoTable1D ht1 ("histo.txt");
  ht1.write(*histo);
  delete histo;
}
int main( int argc, char **argv )
{
  Histo_App app;
```

```
  app.run();
}
```

D.10.2 PAW kumac for reading table

```
mess "Notice that this is an HTL histogram with gravity bins"
mess "The center of each bin is the bin center-of-mass"
mess "rather than the mid point"
* Read x,y,ey; ex are all zeros
ve/read x,y,ey histo.txt
ve/cre ex(20)
zone 2 2
* Draw as histogram
ve/draw y
* Fit with 2nd degree polynomial
ve/fit x y ey p2
* Display as markers+error bars
graphics/hplot/errors x y ex ey 20 25 .07 W
* Finally as a bar chart
graph 20 x y AWB
```

D.10.3 PAW Macro to produce 1D HBOOK histogram from HTL ASCII table

```
* Macro to read an HTL ASCII table an produce a 1D HBOOK histogram
* Assumes binning is fixed and histogram is 1D.
* Since the bin center of HBOOK is always the middle of the bin, you
* may lose some information going from HTL to HBOOK.

Macro htl2hbook 1='Test' 2=1
file  = [1]
id    = [2]
*
ve/read x,y,ey [file]
title  = [file]
nbins  = $vdim(x,1)
width  = x(2)-x(1)
width2 = [width]/2
xmin   = x(1)-[width2]
xmax   = x([nbins])+[width2]
1dhisto [id] [title] [nbins] [xmin] [xmax]
put/cont [id] y
*
v/del x,y,ey
Return
```

D.10.4 Output Generated

```
0 2051.38 43.0276
1 1642.24 34.4459
2 1278.56 26.8177
3 960.343 20.1431
4 687.583 14.422
5 460.282 9.65439
6 278.443 5.84031
7 142.062 2.97975
```

```
8 51.1425 1.07271
9 5.6825 0.11919
10 5.6825 0.11919
11 51.1425 1.07271
12 142.062 2.97975
13 278.443 5.84031
14 460.282 9.65439
15 687.28 14.4188
16 959.92 20.1387
17 1278 26.8118
18 1641.52 34.4383
19 2050.48 43.0181
```

D.11 Histogram naming

The following example demonstrates the usage of histogram naming. Histograms are first created and filled using a histogram factory and then located by name and printed.

D.11.1 Input program

```
/* name_histos.cpp */
#include <iostream.h>
#include <iomanip.h>
#include "HTL/HNaming.h"
class Histo_App : public HepDbApplication
{
public:
  Histo_App ()  {}
  void create();
public:
  void init();
};
void Histo_App::init()
{
  Init();
  startUpdate();
}
void Histo_App::create()
{
  const int noBins=20;
  HNamingFactory myFactory(HNamingFactory::Override);
  // Create histograms in DB "gepo", Container "sbaffini" and name them
  // in /usr/.../Histograms/MC
  if (myFactory.init(this,"Histograms/MC","gepo","sbaffini")) {
    // Create two histograms. They're named 10 and "p2" in the name tree
    HepRef(PHisto1D) h1 = myFactory.Histo1D(10,"Histo-1",noBins,0.0,20.0);
    HepRef(PHisto1D) h2 = myFactory.Histo1D("p2","Histo-2",noBins,0.0,20.0);
    // Fill histos with different weights:
    for( int i = 0; i < 50000; i++) {
      double x = (i % 22) - 1;
      h1->fill(x, (x-9.5)*(x-9.5)+3);
      h2->fill(x, (x-9.5)*(x-9.5)+30);
    }
  }
  // Create histograms in user database, container "Histograms" and name them
  // in /usr/.../Histograms/Data
```

```
  if (myFactory.init(this,"Histograms/Data")) {
    // Create two histograms. They're named 10 and "p2" in the name tree
    HepRef(PHisto1D) h1 = myFactory.Histo1D(10,"Histo-11",noBins,0.0,20.0);
    HepRef(PHisto1D) h2 = myFactory.Histo1D("p2","Histo-22",noBins,0.0,20.0);
    // Fill histos with different weights:
    for( int i = 0; i < 50000; i++) {
      double x = (i % 22) - 1;
      h1->fill(x, (x-9.5)*(x-9.5)+3);
      h2->fill(x, (x-9.5)*(x-9.5)+30);
    }
  }
  // Locate Histogram /usr/.../Histograms/Data/p2
  HLocator myLocator;
  HPrinter hp(cout);
  myLocator.init(this);
  HepRef(PHisto1D) h = myLocator.Histo1D("p2");
  if (h != 0)
    hp.print(*h);
  h = myLocator.Histo1D(10);
  if (h != 0)
    hp.print(*h);
}
int main( int argc, char **argv)
{
  // Create application object:
  Histo_App my_app;
  my_app.init();
  my_app.create();
  my_app.commit();
}
```

D.11.2 Output Generated

```
TYPE       : Histo1D
TITLE      : Histo-2
BIN COUNT  : 20
BIN WIDTH  : 1
                          6.88e+04                         2.73e+05 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     2.733e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+00   1     2.324e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   2     1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     1.642e+05|XXXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     1.369e+05|XXXXXXXXXXXXXXXXX
 5.000e+00   5     1.142e+05|XXXXXXXXXXX
 6.000e+00   6     9.603e+04|XXXXXX
 7.000e+00   7     8.240e+04|XXX
 8.000e+00   8     7.330e+04|X
 9.000e+00   9     6.876e+04|
 1.000e+01   10    6.876e+04|
 1.100e+01   11    7.330e+04|X
 1.200e+01   12    8.240e+04|XXX
 1.300e+01   13    9.603e+04|XXXXXX
 1.400e+01   14    1.142e+05|XXXXXXXXXXX
 1.500e+01   15    1.369e+05|XXXXXXXXXXXXXXXX
 1.600e+01   16    1.642e+05|XXXXXXXXXXXXXXXXXXXXX
 1.700e+01   17    1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    2.323e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
 1.900e+01   19    2.732e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                         X |
                           V
UNDERFLOW :  3.188e+05 OVERFLOW :  3.186e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  6.865e+00
TYPE       : Histo1D
TITLE      : Histo-1
BIN COUNT  : 20
BIN WIDTH  : 1.000e+00
                           7.39e+03                        2.12e+05 Y
X POSITION   BIN       VALUE|------------------------------------------------->
 0.000e+00   0     2.120e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+00   1     1.710e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   2     1.347e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     1.029e+05|XXXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     7.558e+04|XXXXXXXXXXXXXXXX
 5.000e+00   5     5.285e+04|XXXXXXXXXXX
 6.000e+00   6     3.466e+04|XXXXXX
 7.000e+00   7     2.103e+04|XXX
 8.000e+00   8     1.193e+04|X
 9.000e+00   9     7.387e+03|
 1.000e+01   10    7.387e+03|
 1.100e+01   11    1.193e+04|X
 1.200e+01   12    2.103e+04|XXX
 1.300e+01   13    3.466e+04|XXXXXX
 1.400e+01   14    5.285e+04|XXXXXXXXXXX
 1.500e+01   15    7.554e+04|XXXXXXXXXXXXXXX
 1.600e+01   16    1.028e+05|XXXXXXXXXXXXXXXXXXXXX
 1.700e+01   17    1.346e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    1.710e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   19    2.119e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                         X |
                           V
UNDERFLOW :  2.574e+05 OVERFLOW :  2.573e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  7.580e+00
```

D.12 Basic histogram operations

The following example shows a number of basic histogram operations, such as subtraction, multiplication etc.

D.12.1 Input program

```
/* basic_operations.cpp */
/*****************************************************************************
Demonstrate:
. how to create, fill and print a basic Histo1D
. how to create/clone a new histo out of an existing one.
  (constructor and cloning techniques demonstrated)
. how to perform basic operations.
*****************************************************************************/
#include <iostream.h>
#include <iomanip.h>                 // Formatting output string.
#include "HTL/PHistograms.h" // Persistent histograms.
class Histo_App : public HepDbApplication
```

```
{
public:
  Histo_App( const char *a_db_name = 0 )
  { db_name_ = ( a_db_name == 0 ) ? "Default_DB" : a_db_name; }
  // If no database name is provided then the default one is
  // "Default_DB".
public:
  void run();
  void init()
  {
    Init();
    startUpdate();
    db_ = db(db_name_);
  }
private:
  ooHandle(ooDBObj) db_; // Handle on the database.
  const char *db_name_; // Name of the database.
};
// Implementation:
//
void Histo_App::run()
{
  const Size bin_count = 20;
  const double x_min = 0.0;
  const double x_max = 20.0;
  // Create a histo 1D using Gravity_Bin and Even_Partition:
  // (The type of the points is double.)
  //
  HepRef(PHisto1D) histo1 = new (db_) PHisto1D( "Histo1: parabolic function", bin_count, x_min, x_max );
  // Let's fill the histo with 50000 points:
  //
  long i;
  double x, w;
  for( i=0; i<50000; i++ ) {
    x = (i % 22) - 1;
    w = (x-9.5)*(x-9.5)+3;
    histo1->fill(x,w);
  }
  // Print:
  //
  HPrinter hp( cout );
  cout << endl << endl; hp.print( *histo1 );
  // Now create a clone of histo1 and print it out:
  //
  HepRef(PHisto1D) histo2 = histo1->clone();
  // Change name.
  histo2->set_name( "Histo2: a clone of Histo1" );
  cout << endl << endl; hp.print( *histo2 );
  // Add histo1 to histo2 and print the result:
  //
  histo2->add( *histo1 );
  histo2->set_name( "Histo2 =  Histo2+Histo1 = 2*Histo1" );
  cout << endl << endl; hp.print( *histo2 );
  //
  // It is also possible to clone and perform an operation on single
  // statement:
  //         histo2 = histo1->clone()->add( histo1 );
  //
  // Now create a new histo out of histo2:
  // ... (So simple, huh?)
  //
```

```
  HepRef(PHisto1D) histo3 = new(db_) PHisto1D( *histo2 );
  // Sub histo1 to histo3 ==> histo3 should hold values of histo1
  // Mul by -1.0 ==> histo3 should now hold the opposite of histo1
  // And print the result:
  //
  histo3->sub( *histo1 );
  histo3->mul( -1.0 );
  histo3->set_name( "Histo3: -(Histo2 - Histo1) = -Histo1" );
  cout << endl << endl; hp.print( *histo3 );
}
int main( int argc, char **argv )
{
  Histo_App app;
  app.init();
  app.run();
  app.commit();
}
```

D.12.2 Output Generated

```
TYPE       : Histo1D
TITLE      : Histo1: parabolic function
BIN COUNT  : 20
BIN WIDTH  : 1
                         7.39e+03                          2.12e+05 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     2.120e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+00   1     1.710e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   2     1.347e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     1.029e+05|XXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     7.558e+04|XXXXXXXXXXXXXXXX
 5.000e+00   5     5.285e+04|XXXXXXXXXX
 6.000e+00   6     3.466e+04|XXXXXX
 7.000e+00   7     2.103e+04|XXX
 8.000e+00   8     1.193e+04|X
 9.000e+00   9     7.387e+03|
 1.000e+01   10    7.387e+03|
 1.100e+01   11    1.193e+04|X
 1.200e+01   12    2.103e+04|XXX
 1.300e+01   13    3.466e+04|XXXXXX
 1.400e+01   14    5.285e+04|XXXXXXXXXX
 1.500e+01   15    7.554e+04|XXXXXXXXXXXXXXX
 1.600e+01   16    1.028e+05|XXXXXXXXXXXXXXXXXXXX
 1.700e+01   17    1.346e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    1.710e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   19    2.119e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                        X |
                          V
UNDERFLOW :  2.574e+05 OVERFLOW :  2.573e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  7.580e+00
TYPE       : Histo1D
TITLE      : Histo2: a clone of Histo1
BIN COUNT  : 20
BIN WIDTH  : 1.000e+00
                         7.39e+03                          2.12e+05 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     2.120e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+00   1     1.710e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
 2.000e+00   2     1.347e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     1.029e+05|XXXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     7.558e+04|XXXXXXXXXXXXXXXX
 5.000e+00   5     5.285e+04|XXXXXXXXXX
 6.000e+00   6     3.466e+04|XXXXXX
 7.000e+00   7     2.103e+04|XXX
 8.000e+00   8     1.193e+04|X
 9.000e+00   9     7.387e+03|
 1.000e+01   10    7.387e+03|
 1.100e+01   11    1.193e+04|X
 1.200e+01   12    2.103e+04|XXX
 1.300e+01   13    3.466e+04|XXXXXX
 1.400e+01   14    5.285e+04|XXXXXXXXXX
 1.500e+01   15    7.554e+04|XXXXXXXXXXXXXXX
 1.600e+01   16    1.028e+05|XXXXXXXXXXXXXXXXXXXXX
 1.700e+01   17    1.346e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    1.710e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   19    2.119e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                        X |
                          V
UNDERFLOW :  2.574e+05 OVERFLOW :  2.573e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  7.580e+00
TYPE       : Histo1D
TITLE      : Histo2 =  Histo2+Histo1 = 2*Histo1
BIN COUNT  : 20
BIN WIDTH  : 1.000e+00
                           1.48e+04                         4.24e+05 Y
X POSITION   BIN       VALUE|--------------------------------------------------->
 0.000e+00   0     4.239e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+00   1     3.421e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   2     2.694e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     2.057e+05|XXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     1.512e+05|XXXXXXXXXXXXXXX
 5.000e+00   5     1.057e+05|XXXXXXXXXX
 6.000e+00   6     6.933e+04|XXXXXX
 7.000e+00   7     4.205e+04|XXX
 8.000e+00   8     2.387e+04|X
 9.000e+00   9     1.477e+04|
 1.000e+01   10    1.477e+04|
 1.100e+01   11    2.387e+04|X
 1.200e+01   12    4.205e+04|XXX
 1.300e+01   13    6.933e+04|XXXXXX
 1.400e+01   14    1.057e+05|XXXXXXXXXX
 1.500e+01   15    1.511e+05|XXXXXXXXXXXXXX
 1.600e+01   16    2.056e+05|XXXXXXXXXXXXXXXXXXX
 1.700e+01   17    2.692e+05|XXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    3.419e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   19    4.237e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                        X |
                          V
UNDERFLOW :  5.148e+05 OVERFLOW :  5.146e+05
 IN RANGE :      90910 EXTRA    :       9090
 MEAN B.C.:  9.499e+00 RMS B.C. :  7.580e+00
TYPE       : Histo1D
TITLE      : Histo3: -(Histo2 - Histo1) = -Histo1
BIN COUNT  : 20
BIN WIDTH  : 1.000e+00
                          -2.12e+05                        -7.39e+03 Y
X POSITION   BIN       VALUE|--------------------------------------------------->
 0.000e+00   0    -2.120e+05|
```

```
 1.000e+00   1    -1.710e+05|XXXXXXXXXX
 2.000e+00   2    -1.347e+05|XXXXXXXXXXXXXXXXXX
 3.000e+00   3    -1.029e+05|XXXXXXXXXXXXXXXXXXXXXXXXX
 4.000e+00   4    -7.558e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 5.000e+00   5    -5.285e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 6.000e+00   6    -3.466e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 7.000e+00   7    -2.103e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 8.000e+00   8    -1.193e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 9.000e+00   9    -7.387e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+01   10   -7.387e+03|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.100e+01   11   -1.193e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.200e+01   12   -2.103e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.300e+01   13   -3.466e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.400e+01   14   -5.285e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.500e+01   15   -7.554e+04|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.600e+01   16   -1.028e+05|XXXXXXXXXXXXXXXXXXXXXXXXX
 1.700e+01   17   -1.346e+05|XXXXXXXXXXXXXXXXXX
 1.800e+01   18   -1.710e+05|XXXXXXXXXX
 1.900e+01   19   -2.119e+05|
                         X |
                           V
UNDERFLOW : -2.574e+05 OVERFLOW : -2.573e+05
 IN RANGE :     136365 EXTRA    :      13635
 MEAN B.C.:  9.499e+00 RMS B.C. :  7.580e+00
```

D.13 Read histograms

The following example shows the creation of 3 persistent histograms and the location of a particle histogram using a database scan.

D.13.1 Input program

```
/* read_histo.cpp */
#include <iostream.h>
#include <iomanip.h>
#include "HTL/PHistograms.h"
class Histo_App : public HepDbApplication
{
public:
  Histo_App( const char *a_db_name = 0 )
  { db_name_ = ( a_db_name != 0 ) ? a_db_name : "Default_DB"; }
  void create();
  // Create 3 histograms.
  void readHisto();
  // Read back the 2nd histogram from the database.
public:
  void init();
  private :
  ooHandle(ooDBObj)  histoDb;
  const char *db_name_;
};
void Histo_App::init()
{
  Init();
  startUpdate();
  histoDb = db(db_name_);
}
```

```
void Histo_App::create()
{
  const int noOfBins=20;
  // Create three histograms with different names:
  HepRef(PHisto1D) h1 = new (histoDb) PHisto1D( "Histo-1", noOfBins, 0.0, 20.0);
  HepRef(PHisto1D) h2 = new (histoDb) PHisto1D( "Histo-2", noOfBins, 0.0, 20.0);
  HepRef(PHisto1D) h3 = new (histoDb) PHisto1D( "Histo-3", noOfBins, 0.0, 20.0);
  // Fill histos with different weights:
  double x;
  for( int i = 0; i < 50000; i++) {
    x = (i % 22) - 1;
    h1->fill(x, (x-9.5)*(x-9.5)+3);
    h2->fill(x, (x-9.5)*(x-9.5)+30);
    h3->fill(x, (x-9.5)*(x-9.5)+300);
  }
}
void Histo_App::readHisto()
{
  ooItr(PHisto1D)  histo1D_itr;
  // Iterator on 1D histo interface.
  // Note:
  // if the actual type of the histogram would not be known,
  // an iterator for the interface type is needed:
  //         ooItr(P_I_Histo_1D)  histo1D_itr;
  // Scan the database for the previous histo interface:
  // (We could restrict our search to a container only)
  histo1D_itr.scan( histoDb );
  cout << endl << "Start scanning. Looking for Histo-2" << endl;
  // Look for Histo 2:
  //
  for( int i = 1; histo1D_itr.next(); i++ ) {
    cout << "Found: " << histo1D_itr->name() << endl;
    if( strcmp(histo1D_itr->name(), "Histo-2") == 0) {
      // Let's print it:
      cout << endl  << endl;
      HPrinter hp(cout);
      hp.print( *histo1D_itr);
      cout << endl;
    }
  }
}
int main( int argc, char **argv)
{
  // Read database name from command line if any:
  const char *a_db_name = 0;
  if( argc > 1 ) a_db_name = argv[1];
  // Create application object:
  Histo_App my_app( a_db_name );
  my_app.init();
  my_app.create();
  my_app.commit();
  my_app.startRead();
  my_app.readHisto();
  my_app.commit();
}
```

D.13.2 Output Generated

```
Start scanning. Looking for Histo-2
```

```
Found: Histo2 =  Histo2+Histo1 = 2*Histo1
Found: Histo2 =  Histo2+Histo1 = 2*Histo1
Found: Histo2 =  Histo2+Histo1 = 2*Histo1
Found: Histo2 =  Histo2+Histo1 = 2*Histo1
Found: Histo2 =  Histo2+Histo1 = 2*Histo1
Found: Histo2 =  Histo2+Histo1 = 2*Histo1
Found: Histo2 =  Histo2+Histo1 = 2*Histo1
Found: Histo_1D
Found: Histo_1D
Found: Histo1: parabolic function
Found: Histo3: -(Histo2 - Histo1) = -Histo1
Found: Histo-1
Found: Histo-2
TYPE       : Histo1D
TITLE      : Histo-2
BIN COUNT  : 20
BIN WIDTH  : 1
                          6.88e+04                          2.73e+05 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     2.733e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+00   1     2.324e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   2     1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     1.642e+05|XXXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     1.369e+05|XXXXXXXXXXXXXXXXX
 5.000e+00   5     1.142e+05|XXXXXXXXXXX
 6.000e+00   6     9.603e+04|XXXXXX
 7.000e+00   7     8.240e+04|XXX
 8.000e+00   8     7.330e+04|X
 9.000e+00   9     6.876e+04|
 1.000e+01   10    6.876e+04|
 1.100e+01   11    7.330e+04|X
 1.200e+01   12    8.240e+04|XXX
 1.300e+01   13    9.603e+04|XXXXXX
 1.400e+01   14    1.142e+05|XXXXXXXXXXX
 1.500e+01   15    1.369e+05|XXXXXXXXXXXXXXXX
 1.600e+01   16    1.642e+05|XXXXXXXXXXXXXXXXXXXXX
 1.700e+01   17    1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    2.323e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   19    2.732e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                        X |
                          V
UNDERFLOW :  3.188e+05 OVERFLOW :  3.186e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  6.865e+00
Found: Histo-3
Found: Histo_1D
Found: Histo1: parabolic function
Found: Histo3: -(Histo2 - Histo1) = -Histo1
Found: Histo-1
Found: Histo-2
TYPE       : Histo1D
TITLE      : Histo-2
BIN COUNT  : 20
BIN WIDTH  : 1.000e+00
                          6.88e+04                          2.73e+05 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     2.733e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+00   1     2.324e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   2     1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     1.642e+05|XXXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     1.369e+05|XXXXXXXXXXXXXXXXX
```

```
 5.000e+00   5     1.142e+05|XXXXXXXXXXX
 6.000e+00   6     9.603e+04|XXXXXX
 7.000e+00   7     8.240e+04|XXX
 8.000e+00   8     7.330e+04|X
 9.000e+00   9     6.876e+04|
 1.000e+01   10    6.876e+04|
 1.100e+01   11    7.330e+04|X
 1.200e+01   12    8.240e+04|XXX
 1.300e+01   13    9.603e+04|XXXXXX
 1.400e+01   14    1.142e+05|XXXXXXXXXXX
 1.500e+01   15    1.369e+05|XXXXXXXXXXXXXXX
 1.600e+01   16    1.642e+05|XXXXXXXXXXXXXXXXXXXXX
 1.700e+01   17    1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    2.323e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   19    2.732e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                        X |
                          V
UNDERFLOW :  3.188e+05 OVERFLOW :  3.186e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  6.865e+00
Found: Histo-3
Found: Histo_1D
Found: Histo1: parabolic function
Found: Histo3: -(Histo2 - Histo1) = -Histo1
Found: Histo-1
Found: Histo-2
TYPE       : Histo1D
TITLE      : Histo-2
BIN COUNT  : 20
BIN WIDTH  : 1.000e+00
                          6.88e+04                       2.73e+05 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     2.733e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+00   1     2.324e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   2     1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     1.642e+05|XXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     1.369e+05|XXXXXXXXXXXXXXX
 5.000e+00   5     1.142e+05|XXXXXXXXXX
 6.000e+00   6     9.603e+04|XXXXX
 7.000e+00   7     8.240e+04|XXX
 8.000e+00   8     7.330e+04|X
 9.000e+00   9     6.876e+04|
 1.000e+01   10    6.876e+04|
 1.100e+01   11    7.330e+04|X
 1.200e+01   12    8.240e+04|XXX
 1.300e+01   13    9.603e+04|XXXXX
 1.400e+01   14    1.142e+05|XXXXXXXXXX
 1.500e+01   15    1.369e+05|XXXXXXXXXXXXXX
 1.600e+01   16    1.642e+05|XXXXXXXXXXXXXXXXXXX
 1.700e+01   17    1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    2.323e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   19    2.732e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                        X |
                          V
UNDERFLOW :  3.188e+05 OVERFLOW :  3.186e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  6.865e+00
Found: Histo-3
Found: Histo_1D
Found: Histo1: parabolic function
Found: Histo3: -(Histo2 - Histo1) = -Histo1
```

```
Found: Histo-1
Found: Histo-2
TYPE       : Histo1D
TITLE      : Histo-2
BIN COUNT  : 20
BIN WIDTH  : 1.000e+00
                          6.88e+04                          2.73e+05 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     2.733e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+00   1     2.324e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   2     1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     1.642e+05|XXXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     1.369e+05|XXXXXXXXXXXXXXXXX
 5.000e+00   5     1.142e+05|XXXXXXXXXXX
 6.000e+00   6     9.603e+04|XXXXXX
 7.000e+00   7     8.240e+04|XXX
 8.000e+00   8     7.330e+04|X
 9.000e+00   9     6.876e+04|
 1.000e+01   10    6.876e+04|
 1.100e+01   11    7.330e+04|X
 1.200e+01   12    8.240e+04|XXX
 1.300e+01   13    9.603e+04|XXXXXX
 1.400e+01   14    1.142e+05|XXXXXXXXXXX
 1.500e+01   15    1.369e+05|XXXXXXXXXXXXXXX
 1.600e+01   16    1.642e+05|XXXXXXXXXXXXXXXXXXXXX
 1.700e+01   17    1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    2.323e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   19    2.732e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                        X |
                          V
UNDERFLOW :  3.188e+05 OVERFLOW :  3.186e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  6.865e+00
Found: Histo-3
Found: Histo_1D
Found: Histo1: parabolic function
Found: Histo3: -(Histo2 - Histo1) = -Histo1
Found: Histo-1
Found: Histo-2
TYPE       : Histo1D
TITLE      : Histo-2
BIN COUNT  : 20
BIN WIDTH  : 1.000e+00
                          6.88e+04                          2.73e+05 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     2.733e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+00   1     2.324e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   2     1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     1.642e+05|XXXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     1.369e+05|XXXXXXXXXXXXXXXXX
 5.000e+00   5     1.142e+05|XXXXXXXXXXX
 6.000e+00   6     9.603e+04|XXXXXX
 7.000e+00   7     8.240e+04|XXX
 8.000e+00   8     7.330e+04|X
 9.000e+00   9     6.876e+04|
 1.000e+01   10    6.876e+04|
 1.100e+01   11    7.330e+04|X
 1.200e+01   12    8.240e+04|XXX
 1.300e+01   13    9.603e+04|XXXXXX
 1.400e+01   14    1.142e+05|XXXXXXXXXXX
 1.500e+01   15    1.369e+05|XXXXXXXXXXXXXXX
```

```
 1.600e+01   16    1.642e+05|XXXXXXXXXXXXXXXXXXXXX
 1.700e+01   17    1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    2.323e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   19    2.732e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                         X |
                           V
UNDERFLOW :  3.188e+05 OVERFLOW :  3.186e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  6.865e+00
Found: Histo-3
Found: Histo_1D
Found: Histo_1D
Found: Histo_1D
Found: Histo1: parabolic function
Found: Histo3: -(Histo2 - Histo1) = -Histo1
Found: Histo-1
Found: Histo-2
TYPE       : Histo1D
TITLE      : Histo-2
BIN COUNT  : 20
BIN WIDTH  : 1.000e+00
                           6.88e+04                          2.73e+05 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     2.733e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+00   1     2.324e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   2     1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     1.642e+05|XXXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     1.369e+05|XXXXXXXXXXXXXXXX
 5.000e+00   5     1.142e+05|XXXXXXXXXX
 6.000e+00   6     9.603e+04|XXXXX
 7.000e+00   7     8.240e+04|XX
 8.000e+00   8     7.330e+04|X
 9.000e+00   9     6.876e+04|
 1.000e+01   10    6.876e+04|
 1.100e+01   11    7.330e+04|X
 1.200e+01   12    8.240e+04|XX
 1.300e+01   13    9.603e+04|XXXXX
 1.400e+01   14    1.142e+05|XXXXXXXXXX
 1.500e+01   15    1.369e+05|XXXXXXXXXXXXXXX
 1.600e+01   16    1.642e+05|XXXXXXXXXXXXXXXXXXXX
 1.700e+01   17    1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    2.323e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   19    2.732e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                         X |
                           V
UNDERFLOW :  3.188e+05 OVERFLOW :  3.186e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  6.865e+00
Found: Histo-3
Found: Histo_1D
Found: Histo1: parabolic function
Found: Histo3: -(Histo2 - Histo1) = -Histo1
Found: Histo-1
Found: Histo-2
TYPE       : Histo1D
TITLE      : Histo-2
BIN COUNT  : 20
BIN WIDTH  : 1.000e+00
                           6.88e+04                          2.73e+05 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     2.733e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
 1.000e+00   1     2.324e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   2     1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     1.642e+05|XXXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     1.369e+05|XXXXXXXXXXXXXXXX
 5.000e+00   5     1.142e+05|XXXXXXXXXX
 6.000e+00   6     9.603e+04|XXXXXX
 7.000e+00   7     8.240e+04|XXX
 8.000e+00   8     7.330e+04|X
 9.000e+00   9     6.876e+04|
 1.000e+01   10    6.876e+04|
 1.100e+01   11    7.330e+04|X
 1.200e+01   12    8.240e+04|XXX
 1.300e+01   13    9.603e+04|XXXXXX
 1.400e+01   14    1.142e+05|XXXXXXXXXX
 1.500e+01   15    1.369e+05|XXXXXXXXXXXXXXXX
 1.600e+01   16    1.642e+05|XXXXXXXXXXXXXXXXXXXXX
 1.700e+01   17    1.960e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    2.323e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   19    2.732e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                        X |
                          V
UNDERFLOW :  3.188e+05 OVERFLOW :  3.186e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  6.865e+00
Found: Histo-3
```

D.14 Create transient histograms

The following example shows the use of transient histograms.

D.14.1 Input program

```
/* create_transient.cpp */
#include <iostream.h>
#include <iomanip.h>                  // Formatting output string.
#include "HTL/Histograms.h" // Transient histograms.
class Histo_App
{
public:
  Histo_App() {}
public:
  void run();
  // Create 1D & 2D transient histograms and fill them
  // Print simple properties and global statistics.
private:
  void create1D ();
  void create2D ();
};
//
// Implementation:
//
void Histo_App::run() {
  create1D();
  create2D();
}
void Histo_App::create1D() {
  // Create a histo 1D
  Histo1D *histo = new Histo1D("Histo_1D", 20, 0.0, 20.0 );
```

```
  // Let's fill the histo with 50000 points:
  //
  long i;
  double x, w;
  for( i=0; i<50000; i++ ) {
    x = (i % 22) - 1;
    w = (x-9.5)*(x-9.5)+3;
    histo->fill(x,w);
  }
  // Let's print some properties of the new histo:
  //
  cout << "Histo name: " << histo->name() << endl
       << "Bin count : " << histo->bin_count()
       << " from " << histo->partition().lower_point()
       << " to " << histo->partition().upper_point() << endl << endl;
  // Now display some statistics:
  //
  long in_entries = HStat::in_range_entries_count(*histo);
  long extra_entries = HStat::extra_entries_count(*histo);
  cout << "Entries count: " << (in_entries + extra_entries)
       << " with IN_RANGE: " << in_entries
       << " EXTRA: " << extra_entries << endl;
  //double mean = HStat::mean( *histo );
  //cout << "MEAN (B.C.): " << mean << endl;
  //cout << "RMS  (B.C.): " << HStat::rms(*histo, mean)
  //        << endl << endl;
  cout << "MEAN (B.C.): " << histo->mean() << endl;
  cout << "RMS  (B.C.): " << histo->rms() << endl;
  // Print histo contents:
  //
  cout << ". UNDERFLOW: " << histo->extra_bin(H_UNDERFLOW).value();
  for( i=0; i<histo->bin_count(); i++ ) {
    cout << endl << ". " << setw(2) << i << ": "
         << setw(9) << histo->i_bin(i).value() << " +/- "
         << setw(4) << histo->i_bin(i).error();
  }
  cout << endl << ". OVERFLOW: " << histo->extra_bin(H_OVERFLOW).value();
  cout << endl <<  "TOTAL IN_RANGE: "
       << HInfo::in_range_value( *histo ) << endl << endl;
  // Print histo with existing HPrinter class from the library:
  //
  HPrinter hp( cout );
  hp.print( *histo );
  delete histo;
}
void Histo_App::create2D()
{
  const int n_bin = 10;
  // Create a 2D histo
  //
  Histo2D *histo = new Histo2D( "Histo_2D", n_bin, 5., 15., n_bin, 5., 15. );
  // Let's fill the histo with 50000 points:
  //
  double x, w = 0.5;
  for( long i=0; i<50000; i++ ) {
    x = -i*sin(float(i));
    histo->fill(x,x,w);
    histo->fill(20.-x,x,w);
  }
  // Let's print some properties of the new histo:
  //
```

```
  cout << "Histo name: " << histo->name() << endl
       << "X: Bin count : " << histo->partition_X().bin_count()
       << " from " << histo->partition_X().lower_point()
       << " to "   << histo->partition_X().upper_point() << endl;
  cout << "Y: Bin count : " << histo->partition_Y().bin_count()
       << " from " << histo->partition_Y().lower_point()
       << " to "   << histo->partition_Y().upper_point() << endl;
  // Now some statistics:
  //
  long in_entries = HStat::in_range_entries_count( *histo );
  long extra_entries = HStat::extra_entries_count( *histo );
  cout << "Entries count: " << (in_entries + extra_entries)
       << " with IN_RANGE: " << in_entries
       << " EXTRA: " << extra_entries << endl;
  cout << "In Range values: " << HInfo::in_range_value( *histo )
       << endl;
  // Print histo contents:
  //
#define EH(h,i,j)         h->extra_bin(i,j).value()
  cout << endl;
  cout << "(U,O): " << EH( histo, H_UNDERFLOW, H_OVERFLOW )
       << " (I,O): " << EH( histo, H_IN_RANGE, H_OVERFLOW )
       << " (O,O): " << EH( histo, H_OVERFLOW, H_OVERFLOW ) << endl;
  cout << "(U,I): " << EH( histo, H_UNDERFLOW, H_IN_RANGE )
       << " (I,I): " << "XXXXXX"
       << " (O,I): " << EH( histo, H_OVERFLOW, H_IN_RANGE ) <<endl;
  cout << "(U,U): " << EH( histo, H_UNDERFLOW, H_UNDERFLOW )
       << " (I,U): " << EH( histo, H_IN_RANGE, H_UNDERFLOW )
       << " (O,U): " << EH( histo, H_OVERFLOW, H_UNDERFLOW ) << endl;
  cout << endl;
  for( long y=0; y<histo->partition_Y().bin_count(); y++ )
    for( long x=0; x<histo->partition_X().bin_count(); x++ ) {
      if( histo->bin(x,y).value() != 0.0 ) {
        cout << "bin(" << x << "," << y << "): "
             << histo->bin(x,y).value() << " -/+ "
             << histo->bin(x,y).error() << endl;
      }
    }
  cout << endl;
  // Print using HPrinter:
  //
  HPrinter hp( cout );
  hp.print( *histo );
  delete histo;
}
int main( int argc, char **argv )
{
  Histo_App app;
  app.run();
}
```

D.14.2 Output Generated

```
Histo name: Histo_1D
Bin count : 20 from 0 to 20
Entries count: 50000 with IN_RANGE: 45455 EXTRA: 4545
MEAN (B.C.): 9.49851
RMS  (B.C.): 7.58048
. UNDERFLOW: 257417
```

```
.  0:    211957 +/- 4445.79
.  1:    171043 +/- 3587.62
.  2:    134675 +/- 2824.8
.  3:    102853 +/- 2157.34
.  4:   75577.2 +/- 1585.23
.  5:   52847.2 +/- 1108.47
.  6:   34663.2 +/- 727.059
.  7:   21025.2 +/- 441.003
.  8:   11933.2 +/- 250.299
.  9:   7387.25 +/- 154.947
. 10:   7387.25 +/- 154.947
. 11:   11933.2 +/- 250.299
. 12:   21025.2 +/- 441.003
. 13:   34663.2 +/- 727.059
. 14:   52847.2 +/- 1108.47
. 15:     75544 +/- 1584.88
. 16:    102808 +/- 2156.86
. 17:    134616 +/- 2824.18
. 18:    170968 +/- 3586.83
. 19:    211864 +/- 4444.81
. OVERFLOW: 257304
TOTAL IN_RANGE: 1.64762e+06
TYPE       : Histo1D
TITLE      : Histo_1D
BIN COUNT  : 20
BIN WIDTH  : 1
                        7.39e+03                          2.12e+05 Y
X POSITION   BIN       VALUE|-------------------------------------------------->
 0.000e+00   0     2.120e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.000e+00   1     1.710e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 2.000e+00   2     1.347e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3.000e+00   3     1.029e+05|XXXXXXXXXXXXXXXXXXXX
 4.000e+00   4     7.558e+04|XXXXXXXXXXXXXXX
 5.000e+00   5     5.285e+04|XXXXXXXXXX
 6.000e+00   6     3.466e+04|XXXXXX
 7.000e+00   7     2.103e+04|XXX
 8.000e+00   8     1.193e+04|X
 9.000e+00   9     7.387e+03|
 1.000e+01   10    7.387e+03|
 1.100e+01   11    1.193e+04|X
 1.200e+01   12    2.103e+04|XXX
 1.300e+01   13    3.466e+04|XXXXXX
 1.400e+01   14    5.285e+04|XXXXXXXXXX
 1.500e+01   15    7.554e+04|XXXXXXXXXXXXXXX
 1.600e+01   16    1.028e+05|XXXXXXXXXXXXXXXXXXXX
 1.700e+01   17    1.346e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.800e+01   18    1.710e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 1.900e+01   19    2.119e+05|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                        X |
                          V
UNDERFLOW :  2.574e+05 OVERFLOW :  2.573e+05
 IN RANGE :      45455 EXTRA    :       4545
 MEAN B.C.:  9.499e+00 RMS B.C. :  7.580e+00
Histo name: Histo_2D
X: Bin count : 10 from 5.000e+00 to 1.500e+01
Y: Bin count : 10 from 5.000e+00 to 1.500e+01
Entries count: 100000 with IN_RANGE: 58 EXTRA: 99942
In Range values: 2.900e+01
(U,O): 1.248e+04 (I,O): 0.000e+00 (O,O): 1.248e+04
(U,I): 0.000e+00 (I,I): XXXXXX (O,I): 0.000e+00
(U,U): 1.251e+04 (I,U): 0.000e+00 (O,U): 1.251e+04
```

```
bin(0,0): 1.000e+00 -/+ 7.071e-01
bin(9,0): 1.000e+00 -/+ 7.071e-01
bin(1,1): 2.000e+00 -/+ 1.000e+00
bin(8,1): 2.000e+00 -/+ 1.000e+00
bin(2,2): 2.000e+00 -/+ 1.000e+00
bin(7,2): 2.000e+00 -/+ 1.000e+00
bin(3,3): 5.000e-01 -/+ 5.000e-01
bin(6,3): 5.000e-01 -/+ 5.000e-01
bin(4,4): 1.000e+00 -/+ 7.071e-01
bin(5,4): 1.000e+00 -/+ 7.071e-01
bin(4,5): 2.000e+00 -/+ 1.000e+00
bin(5,5): 2.000e+00 -/+ 1.000e+00
bin(3,6): 1.000e+00 -/+ 7.071e-01
bin(6,6): 1.000e+00 -/+ 7.071e-01
bin(2,7): 2.000e+00 -/+ 1.000e+00
bin(7,7): 2.000e+00 -/+ 1.000e+00
bin(1,8): 1.500e+00 -/+ 8.660e-01
bin(8,8): 1.500e+00 -/+ 8.660e-01
bin(0,9): 1.500e+00 -/+ 8.660e-01
bin(9,9): 1.500e+00 -/+ 8.660e-01
TYPE     : Histo2D
TITLE    : Histo_2D
X:  BINS :    10  WIDTH  : 1.000e+00  MIN    : 5.000e+00  MAX    : 1.500e+01
Y:  BINS :    10  WIDTH  : 1.000e+00  MIN    : 5.000e+00  MAX    : 1.500e+01
         0
         0123456789
        ************
       *            *
   0   *  7........7  *
   1   *  .F......F.  *
   2   *  ..F....F..  *
   3   *  ...3..3...  *
   4   *  ....77....  *
   5   *  ....FF....  *
   6   *  ...7..7...  *
   7   *  ..F....F..  *
   8   *  .B......B.  *
   9   *  B........B  *
       *            *
        ************
ENTRIES  :     100000   Z MIN    :          0
TOTAL C. :         29   Z STEP   :      0.133
Z SCALE  : .+23456789ABCDE
      1.25e+04 |          0 |   1.25e+04
    -----------|------------|-----------
             0 |         29 |          0
    -----------|------------|-----------
      1.25e+04 |          0 |   1.25e+04
```

D.15 Create sliced projections

D.15.1 Input program

```
/* slice_proj.cpp */
#include <iostream.h>
#include <iomanip.h>                 // Formatting output string.
#include "HTL/Histograms.h" // Transient histograms.
class Histo_App
```

```
{
public:
  Histo_App() {}
public:
  void run();
private:
  void slice ();
  void project ();
  Histo2D *histo;
};
//
// Implementation:
//
void Histo_App::run() {
  // Create a 2D histo using Weighted_Bin and Even_Partition:
  histo = new Histo2D( "Histo_2D",10, 5., 15., 10, 5., 15. );
  // Let's fill the histo with 50000 points:
  double x, w = 0.5;
  for( long i=0; i<50000; i++ ) {
    x = -i*sin(float(i));
    histo->fill(x,x,w);
    histo->fill(20.-x,x,w);
  }
  cout << "2D histogram content" << endl;
  for( long y=0; y<histo->partition_Y().bin_count(); y++ )
    for( long x=0; x<histo->partition_X().bin_count(); x++ ) {
        cout << "bin(" << x << "," << y << "): "
             << histo->bin(x,y).value() << " -/+ "
             << histo->bin(x,y).error() << endl;
    }
  // Make two slices
  slice();
  // and two projections
  project();
  delete histo;
}
void Histo_App::slice() {
  long x;
  HSlicer mySlicer;
  // First slice in X . The 2nd parameter is the bin no. along y (0->first bin)
  Histo1DVar *slice1 = mySlicer.xBand (*histo,0);
  // First slice in Y . The 2nd parameter is the x coordinate
  Histo1DVar *slice2 = mySlicer.yBand (*histo,5.);
  cout << endl << "First slice";
  for( x=0; x<slice1->bin_count(); x++ ) {
    cout << endl << ". " << setw(2) << x << ": "
         << setw(9) << slice1->i_bin(x).value() << " +/- "
         << setw(4) << slice1->i_bin(x).error();
  }
  cout << endl << endl << "Second slice";
  for( x=0; x<slice2->bin_count(); x++ ) {
    cout << endl << ". " << setw(2) << x << ": "
         << setw(9) << slice2->i_bin(x).value() << " +/- "
         << setw(4) << slice2->i_bin(x).error();
  }
  delete slice1;
  delete slice2;
}
void Histo_App::project() {
  long x;
  HSlicer myProjector;
```

```
  Histo1DVar *proj1 = myProjector.xProject (*histo);
  Histo1DVar *proj2 = myProjector.yProject (*histo);
  cout << endl << endl << "First projection";
  for( x=0; x<proj1->bin_count(); x++ ) {
    cout << endl << ". " << setw(2) << x << ": "
         << setw(9) << proj1->i_bin(x).value() << " +/- "
         << setw(4) << proj1->i_bin(x).error();
  }
  cout << endl << endl << "Second projection";
  for( x=0; x<proj2->bin_count(); x++ ) {
    cout << endl << ". " << setw(2) << x << ": "
         << setw(9) << proj2->i_bin(x).value() << " +/- "
         << setw(4) << proj2->i_bin(x).error();
  }
  cout << endl;
  delete proj1;
  delete proj2;
}
int main( int argc, char **argv )
{
  Histo_App app;
  app.run();
}
```

D.15.2 Output Generated

```
2D histogram content
bin(0,0): 1 -/+ 0.707107
bin(1,0): 0 -/+ 0
bin(2,0): 0 -/+ 0
bin(3,0): 0 -/+ 0
bin(4,0): 0 -/+ 0
bin(5,0): 0 -/+ 0
bin(6,0): 0 -/+ 0
bin(7,0): 0 -/+ 0
bin(8,0): 0 -/+ 0
bin(9,0): 1 -/+ 0.707107
bin(0,1): 0 -/+ 0
bin(1,1): 2 -/+ 1
bin(2,1): 0 -/+ 0
bin(3,1): 0 -/+ 0
bin(4,1): 0 -/+ 0
bin(5,1): 0 -/+ 0
bin(6,1): 0 -/+ 0
bin(7,1): 0 -/+ 0
bin(8,1): 2 -/+ 1
bin(9,1): 0 -/+ 0
bin(0,2): 0 -/+ 0
bin(1,2): 0 -/+ 0
bin(2,2): 2 -/+ 1
bin(3,2): 0 -/+ 0
bin(4,2): 0 -/+ 0
bin(5,2): 0 -/+ 0
bin(6,2): 0 -/+ 0
bin(7,2): 2 -/+ 1
bin(8,2): 0 -/+ 0
bin(9,2): 0 -/+ 0
bin(0,3): 0 -/+ 0
bin(1,3): 0 -/+ 0
```

```
bin(2,3): 0 -/+ 0
bin(3,3): 0.5 -/+ 0.5
bin(4,3): 0 -/+ 0
bin(5,3): 0 -/+ 0
bin(6,3): 0.5 -/+ 0.5
bin(7,3): 0 -/+ 0
bin(8,3): 0 -/+ 0
bin(9,3): 0 -/+ 0
bin(0,4): 0 -/+ 0
bin(1,4): 0 -/+ 0
bin(2,4): 0 -/+ 0
bin(3,4): 0 -/+ 0
bin(4,4): 1 -/+ 0.707107
bin(5,4): 1 -/+ 0.707107
bin(6,4): 0 -/+ 0
bin(7,4): 0 -/+ 0
bin(8,4): 0 -/+ 0
bin(9,4): 0 -/+ 0
bin(0,5): 0 -/+ 0
bin(1,5): 0 -/+ 0
bin(2,5): 0 -/+ 0
bin(3,5): 0 -/+ 0
bin(4,5): 2 -/+ 1
bin(5,5): 2 -/+ 1
bin(6,5): 0 -/+ 0
bin(7,5): 0 -/+ 0
bin(8,5): 0 -/+ 0
bin(9,5): 0 -/+ 0
bin(0,6): 0 -/+ 0
bin(1,6): 0 -/+ 0
bin(2,6): 0 -/+ 0
bin(3,6): 1 -/+ 0.707107
bin(4,6): 0 -/+ 0
bin(5,6): 0 -/+ 0
bin(6,6): 1 -/+ 0.707107
bin(7,6): 0 -/+ 0
bin(8,6): 0 -/+ 0
bin(9,6): 0 -/+ 0
bin(0,7): 0 -/+ 0
bin(1,7): 0 -/+ 0
bin(2,7): 2 -/+ 1
bin(3,7): 0 -/+ 0
bin(4,7): 0 -/+ 0
bin(5,7): 0 -/+ 0
bin(6,7): 0 -/+ 0
bin(7,7): 2 -/+ 1
bin(8,7): 0 -/+ 0
bin(9,7): 0 -/+ 0
bin(0,8): 0 -/+ 0
bin(1,8): 1.5 -/+ 0.866025
bin(2,8): 0 -/+ 0
bin(3,8): 0 -/+ 0
bin(4,8): 0 -/+ 0
bin(5,8): 0 -/+ 0
bin(6,8): 0 -/+ 0
bin(7,8): 0 -/+ 0
bin(8,8): 1.5 -/+ 0.866025
bin(9,8): 0 -/+ 0
bin(0,9): 1.5 -/+ 0.866025
bin(1,9): 0 -/+ 0
bin(2,9): 0 -/+ 0
```

```
bin(3,9): 0 -/+ 0
bin(4,9): 0 -/+ 0
bin(5,9): 0 -/+ 0
bin(6,9): 0 -/+ 0
bin(7,9): 0 -/+ 0
bin(8,9): 0 -/+ 0
bin(9,9): 1.5 -/+ 0.866025
First slice
.  0:         1 +/- 0.707107
.  1:         0 +/-    0
.  2:         0 +/-    0
.  3:         0 +/-    0
.  4:         0 +/-    0
.  5:         0 +/-    0
.  6:         0 +/-    0
.  7:         0 +/-    0
.  8:         0 +/-    0
.  9:         1 +/- 0.707107
Second slice
.  0:         1 +/- 0.707107
.  1:         0 +/-    0
.  2:         0 +/-    0
.  3:         0 +/-    0
.  4:         0 +/-    0
.  5:         0 +/-    0
.  6:         0 +/-    0
.  7:         0 +/-    0
.  8:         0 +/-    0
.  9:       1.5 +/- 0.866025
First projection
.  0:         2 +/-    1
.  1:         4 +/- 1.41421
.  2:         4 +/- 1.41421
.  3:         1 +/- 0.707107
.  4:         2 +/-    1
.  5:         4 +/- 1.41421
.  6:         2 +/-    1
.  7:         4 +/- 1.41421
.  8:         3 +/- 1.22474
.  9:         3 +/- 1.22474
Second projection
.  0:       2.5 +/- 1.11803
.  1:       3.5 +/- 1.32288
.  2:         4 +/- 1.41421
.  3:       1.5 +/- 0.866025
.  4:         3 +/- 1.22474
.  5:         3 +/- 1.22474
.  6:       1.5 +/- 0.866025
.  7:         4 +/- 1.41421
.  8:       3.5 +/- 1.32288
.  9:       2.5 +/- 1.11803
```