

Histogram DB for Online Monitoring – User's Manual

Issue: 1
Revision: 1

Reference:
Created: January 12, 2007
Last modified: January 18, 2007

Prepared by: G. Graziani

1 DB design

The requirements and use cases of an Histogram Database for Online Monitoring in the context of a common Histogramming Framework [1] have been defined in [2].

The present design of DB tables is shown in figure 1

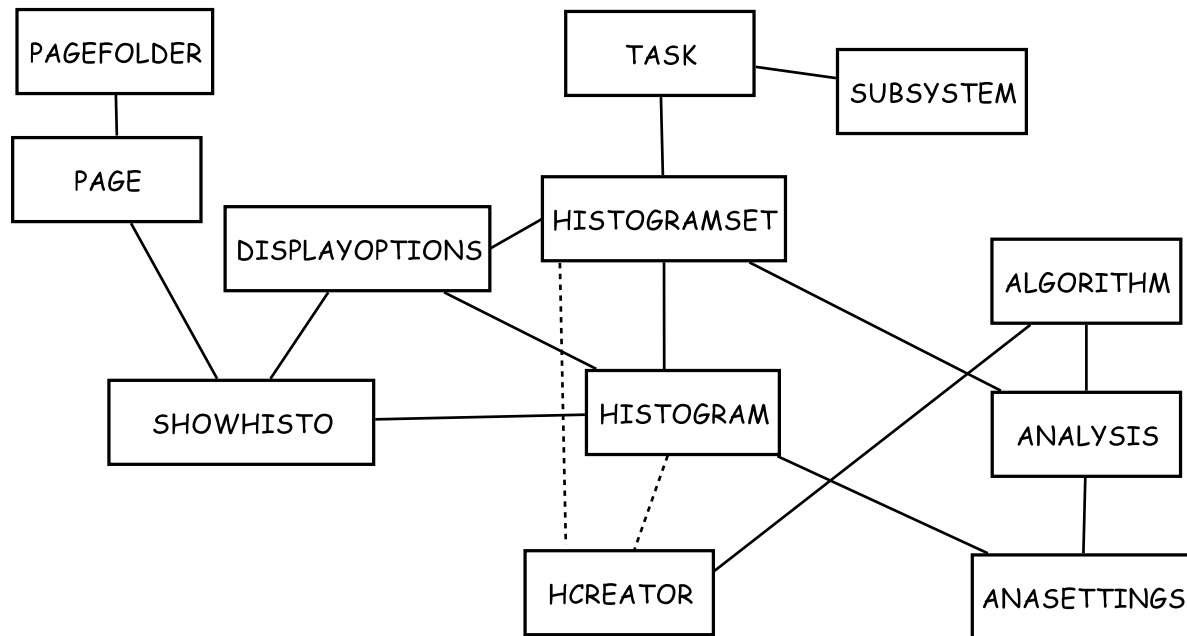


Figure 1 Scheme of DB tables.

1.1 Definition of Histograms

Histograms are uniquely identified by their name:

Taskname / Algorithmname / HistogramTitle

The histogram title can have a subtitle: *HistogramTitle = HistogramSetTitle_\$\$_Subtitle*

Histograms differing only by *Subtitle* are part of the same Histogram set. These should be histograms that have identical binning, e.g. containing the same distribution for different channels of a detector.

For easier reference, an internal unique identifier is created for each histogram, in the form

HID = HSID / IHS

where *HSID* is an integer number identifying the histogram set, and *IHS* is a sequence number (starting from 1) to identify histograms in the same set.

1.1.1 Properties of TASK

- TaskName (string of max length 100)
unique task identifier
- RunOnPhysics, RunOnCalib, RunOnEmpty (boolean)
specify for which type of data task is running
- Subsys1, Subsys2, Subsys3 (string of length 10)
up to 3 subdetector/subsystem can be associated to task.
- Reference (string of length 100)
link to the location of reference histograms for this task

1.1.2 Properties of HISTOGRAMSET

- HSID (integer)
- NHS (integer)
number of histograms in set
- Task (valid TaskName)
- Algorithm (string of max length 100)
- HistogramSetTitle (string of max length 200)
- Type
'1D', '2D', or 'PROF' for profile histogram
- Nanalysis (integer)
number of analysis to be performed on set
- Description (string of max length 4000)
- Documentation (string of max length 200)
link to a more extensive documentation
- HSDisplay (valid DOID)
identifier of display option set associated to Histogram set

1.1.3 Properties of HISTOGRAM

- HID (string of max length 12)
HSID/IHS
- Subtitle (string of max length 50)
- DIMServiceName (string of max length 500)
Name of the DIM service that is currently publishing the histogram
- IsAnalysisHist (boolean)
true if histogram is produced at analysis level
- CreationTime (timestamp)
recording the first time the histogram is seen
- ObsolescenceTime (timestamp)
can be set by hand if histogram is not produced any more
- Display (valid DOID) identifier of display option set associated to Histogram

1.2 Definition of Pages and Display Options

Pages and Page Folders are uniquely identified by their name. Pages are associated to a list of valid histograms through the SHOWHISTO table, containing the layout of each histogram on the page.

As shown in figure 1, a set of display options can be defined for:

- an histogram on a given page
- an histogram
- an histogram set

so that the most specific available set is used, but one can use the same default for, say, the 2000 histograms of a certain set.

1.2.1 Properties of *DISPLAYOPTIONS*

- DOID (integer)
unique identifier
- LABEL_X (string of max length 50)
- LABEL_Y (string of max length 50)
- LABEL_Z (string of max length 50)
- YMIN (float)
- YMAX (float)
- STATS (int)
- FILLSTYLE (int)
- FILLCOLOR (int)
- LINESTYLE (int)
- LINECOLOR (int)
- LINEWIDTH (int)
- DRAWOPTS (string of max length 50)

1.2.2 Properties of *PAGEFOLDER*

- PageFolderName (string of max length 30)
unique page folder identifier

1.2.3 Properties of *PAGE*

- PageName (string of max length 50)
unique page identifier
- Folder (valid PageFolderName)
- Nhisto (integer)
number of histograms on page
- PageDoc (string of max length 100)
short page description

1.2.4 Properties of *SHOWHISTO*

- Page (valid PageName)
- Histo (valid HID)
- Cx, Cy, Sx, Sy (float numbers from 0 to 1)
coordinates of the histogram pad on the page: Cx and Cy define the position of the top left corner, Sx and Sy the size, relatively to the window size
- Sdisplay (valid DOID)
identifier of display option set associated to this Histogram on this Page

1.3 Definition of Automatic Analysis

The ALGORITHM table contains the definition of the algorithms available for analysis. They can be used to create new histograms at analysis level (these will be called "Analysis Histograms" and are defined by the HCREATOR table), or to perform automatic checks, defined in the ANALYSIS table. Analyses are properties of an histogram set, though their parameters can be specified for each histogram in the ANASETTINGS table.

1.3.1 Properties of ALGORITHM

- AlgorithmName (string of max length 30)
unique algorithm identifier
- AlgType
'HCREATOR' or 'CHECK'
- Ninput (integer)
number of input histograms (for 'HCREATOR' algorithms)
- Npars (integer)
number of parameters
- AlgPars (array(any length) of string of max length 15)
parameter names
- AlgDoc (string of max length 1000) documentation

1.3.2 Properties of ANALYSIS

- AID (integer)
unique analysis identifier (allowing to assign the same algorithm more than once to the same histogram)
- HSET (valid HSID)
- Algorithm (valid AlgorithmName)

1.3.3 Properties of ANASETTINGS

- AnaID (valid AID)
- Histogram (valid HID)
- Mask (boolean)
allow to mask the analysis for a single histogram
- Warnings, Alarms (arrays(Npars) of floats)
2 sets of threshold levels

1.3.4 Properties of HCREATOR

when a HCREATOR entry is defined, the corresponding histogram is created with Task='ANALYSIS' and Algorithm= the name of the analysis algorithm

- HCID (valid HID)
- Algorithm (valid AlgorithmName)
- Sourceh (arrays(8) of string of max length 12)
list of input histograms
- SourceHSet (valid HSID)
input histogram set (if required by the algorithm)

2 DB implementation

A first prototype of the DB has been implemented under Oracle on the CERN Oracle server and is available for tests.

The DB can be accessed through a C++ API or interactively through a Web interface written in PHP. In order to minimize client load and network traffic, and ease the maintenance of interface code, both interfaces are based on a set of common PL/SQL procedures that are precompiled on the Oracle server.

3 Web interface

It is available for test at the address

<https://webafs3.cern.ch/ggrazian/lhcb/OnlineHistDB/index.php>

It is intended to be the most suitable tool to browse available histograms, edit the display options and the automatic analysis, including the definition of histograms to be produced at analysis level.

Presently, it is also possible to edit the viewer page configurations, though a graphical editor in the presenter application will likely be the most suitable tool for that task.

4 C++ Interface

The interface is available as a link library that can be compiled from the package

Online/OnlineHistDB

in the LHCb code repository.

The C++ API allows to perform any operation on the DB but the deletion of objects, that is reserved to the DB administrator.

You can add entries to the DB through the methods beginning with *declare*, that create the specified entry if not existing, or update its fields otherwise. Thus, running the same code twice is equivalent to run it once.

4.1 OnlineHistDB class

Each instantiation of this class opens a transaction with the DB server. Its methods allow the creation of histograms and pages, and the definitions of tasks, subsystems and algorithms. Histograms and pages can be edited through the pointers to *OnlineHistogram* and *OnlineHistPage* objects returned by the *getHistogram* and *getPage* methods. Such objects make sense only within the transaction and should never be deleted by the user (they are destroyed by the *OnlineHistDB* destructor).

Changes are committed to the DB only by an explicit call to the *commit* method.

- **HistDB**(std::string passwd, std::string user="lhcb_mon_giacomo", std::string db="devdb10");
constructor
- void **commit**();
commits all changes to the DB
- void **declareTask**(std::string Name, std::string SubDet1="NULL", std::string SubDet2="NULL", std::string SubDet3="NULL", bool RunsOnPhysics=false, bool RunsOnCalib=false, bool RunsOnEmpty=false);
creates or updates a Task definition. Tasks can be associated to up to three subdetectors/subsystems.
- void **declareSubSystem**(std::string SubSys);
declares a subsystem

- **void declareHistByServiceName**(const std::string &ServiceName);
declares an Histogram by its DIM service name. In the LHCb DAQ, this is intended to be used only by the Experiment Control System to dynamically update the DB with the published histograms.
Tasks not known to the DB are automatically created.
if histogram already exists, just updates the current DIM service name
- **void setHistogramBufferDepth**(int N);
when creating histograms with the *declareHistByServiceName* method, the histogram list is actually send to the DB server every N histograms (or at commit) in order to optimize performance. The default buffer depth (recommended value) is 1000.
- **OnlineHistogram* declareAnalysisHistogram**(std::string Algorithm, std::string Title, std::vector<OnlineHistogram*> &Sources);
declares an histogram to be produced at analysis level using algorithm Algorithm. Title is the histogram title. Sources must contain the pointers to the input histograms. If the algorithm requires an histogram set as input, use any histogram of the set. Returns the pointer to the new histogram object.
- **void declareCheckAlgorithm**(std::string Name, int Npars, std::string* pars=NULL, std::string doc="NONE");
declares to the DB an Analysis algorithm implemented in the Analysis library. Npars is the number of algorithm's parameters, pars should point to an array containing the parameter names, doc is a short description of the algorithm.
- **void declareCreatorAlgorithm**(std::string Name, int Ninput=0, std::string doc="NONE");
declares to the DB an available algorithm to produce histograms at analysis time. Ninput is the number of input histograms, doc is a short description of the algorithm.
- **int getAlgorithmNpar**(std::string AlgName);
gets number of parameters needed by algorithm AlgName
- **std::string getAlgParName**(std::string AlgName, int Ipar);
gets the name of parameter Ipar (starting from 1) of algorithm AlgName
- **OnlineHistPage* getPage**(std::string Name, std::string Folder="");
gets a pointer to an OnlineHistPage object, to create a new page (in this case Folder must be specified) or view/edit an existing one
- **OnlineHistogram* getHistogram**(std::string Name, std::string Page="_NONE_");
gets a pointer to an OnlineHistogram object that can be used to view/edit an histogram record. If Page is specified, the default display options for the histogram are those associated to the page (if available).
- **int getHistogramsWithAnalysis**(std::vector<OnlineHistogram*> &list);
gets the list of histograms on which some check analysis has to be performed. Returns the number of histograms found. list must be created by the user.
- **int getAnalysisHistograms**(std::vector<OnlineHistogram*> &list);
gets the list of histograms that have to be produced by analysis task. Returns the number of histograms found. list must be created by the user.
- **int getHistogramsBySubsystem**(std::string SubSys, std::vector<OnlineHistogram*> &list);
gets the list of histograms related to subsystem SubSys. Returns the number of histograms found. list must be created by the user.

- `int getHistogramsByTask(std::string Task, std::vector<OnlineHistogram*> & list);`
gets the list of histograms related to task Task. Returns the number of histograms found. list must be created by the user.
- `int getHistogramsByPage(std::string Page, std::vector<OnlineHistogram*> & list);`
gets the list of histograms displayed on page Page. Returns the number of histograms found. list must be created by the user.
- `int getHistogramsBySet(std::string SetName, std::vector<OnlineHistogram*> & list);`
`int getHistogramsBySet(const OnlineHistogram& Set, std::vector<OnlineHistogram*> & list);`
gets the list of histograms in a Set. Returns the number of histograms found. list must be created by the user.

4.2 OnlineHistogram class

OnlineHistogram objects are instantiated within an OnlineHistDB object, i.e. a DB transaction, through the *getHistogram* method.

- `std::string name();`
full histogram name
- `std::string page();`
page on which histogram is displayed (if specified)
- `void setPage(std::string Page);`
set page on which histogram is displayed (reload display options if needed)
- `std::string hid();`
internal histogram ID (equivalent to `hsid()`/`ihs()`)
- `int hsid();`
internal histogram set ID
- `std::string dimServiceName();`
name of the DIM service that is currently publishing the histogram
- `int nhs();`
number of histograms in set
- `int ihs();`
position of this histogram in set (starting from 1)
- `std::string hstype();`
histogram type ("1D", "2D" or "PROF")
- `std::string hstitle();`
histogram set title
- `std::string subtitle();`
subtitle
- `std::string task();`
task name
- `std::string algo();`
algorithm name

- `std::string descr();`
short description of the histogram
- `std::string doc();`
link to a more extensive documentation
- `int creation();`
creation date, as a unix timestamp
- `int obsoleteness();`
if the histogram is no more in use, returns the end-of-validity date as a unix timestamp, otherwise returns 0.

Methods for display options:

- `typedef enum NONE, SET, HIST, HISTPAGE DisplayOptionMode;`
`DisplayOptionMode domode();`
specifies if the display options in this object are: not defined, associated to the histogram set, associated to the histogram, associated to the histogram on page *page()*
- `bool initDisplayOptionsFromSet();`
initializes display options associated to this histogram with the options defined for the histogram set (if available). Returns true on success.
- `bool initHistoPageDisplayOptionsFromSet(std::string PageName = "_DEFAULT_");`
initializes display options associated to this histogram on page *PageName* (default is *page()*) with the options defined for the histogram set (if available). Returns true on success.
- `bool initHistoPageDisplayOptionsFromHist(std::string PageName = "_NONE_");`
initializes display options associated to this histogram on page *PageName* (default is *page()*) with the options defined for the histogram (if available). Returns true on success.
- `bool setHistoSetDisplayOption(std::string ParameterName, void* value);`
sets a display option for the whole histogram set. The available parameter names and the corresponding types are listed in section 1.2.1. Returns true on success.
- `bool setDisplayOption(std::string ParameterName, void* value);`
sets a display option for the present histogram.
- `bool setHistoPageDisplayOption(std::string ParameterName, void* value, std::string PageName = "_DEFAULT_");`
sets a display option for the present histogram on page *PageName* (default is *page()*)
- `bool getDisplayOption(std::string ParameterName, void* option);`
if display option *ParameterName* has been defined, puts its value into *option* and returns true. In case different sets of display options exists for histogram set, histogram and histogram in page, the most specific one is used.

Methods for analysis options:

- `int nanalysis();`
number of analysis to be performed on the histogram set
- `const std::vector<int>& anaId();`
vector of length *nanalysis()* containing the analysis internal IDs

- `const std::vector<std::string>& anaName();`
vector of length `nanalysis()` containing the analysis algorithm names
- `boolean isAnaHist();`
true if the histogram is produced at analysis level
- `int declareAnalysis(std::string Algorithm, std::vector<float>* warningThr=NULL, std::vector<float>* alarmThr=NULL, int instance=1);`
declare an analysis to be performed on the histogram set. If the algorithm requires some parameters, the warning and alarm values must be specified as vectors of floats and will be set for all histograms in set (then, you can specify values for single histograms with the `setAnalysis` method. You can create more than one analysis with the same algorithm by using `instance>1`. If the analysis identified by Algorithm and instance already exists, parameters are updated. Returns the internal analysis ID.
- `void setAnalysis(int AnaID, std::vector<float>* warningThr = NULL, std::vector<float>* alarmThr = NULL);`
updates parameters for analysis with ID AnaID (for this histogram only)
- `void getAnaSettings(int AnaID, std::vector<float>* warn, std::vector<float>* alarm);`
gets parameters for analysis with ID AnaID
- `void maskAnalysis(int AnalysisAnaID, bool Mask=true);`
masks analysis with ID AnaID. Use `Mask=false` to unmask

4.3 OnlineHistPage class

- `const std::string& name();`
page name
- `const std::string& folder();`
page folder name
- `int nh();`
number of histograms on page
- `const std::string& doc();`
short page description
- `void setDoc(std::string Doc);`
set short page description
- `void setFolder(std::string Folder);`
set page folder name. If not existing, page folder is created.
- `void declareHistogram(OnlineHistogram* h, float Cx, float Cy, float Sx, float Sy, int instance=1);`
adds or updates an histogram on the page. Use `instance>1` to use the same histogram more than once.
- `void removeHistogram(OnlineHistogram* h, unsigned int instance=1);`
remove histogram from page
- `const std::vector<OnlineHistogram*>& hlist();`
returns the list of histograms on this page

- `bool getHistLayout(OnlineHistogram* h, float &Cx, float &Cy, float &Sx, float &Sy, unsigned int instance=1);`
get the layout of given histogram
- `void save();`
actually sends to DB the present page layout. This is automatically called by *OnlineHistDB::commit*, but you may need to call it explicitly if you want to update page-specific histogram options before commit

5 Code Example

```
#include <OnlineHistDB/OnlineHistDB.h>
int main ()
{
```

Open DB transaction:

```
std::string password="ask_to_DB_manager";
OnlineHistDB *HistDB = new OnlineHistDB(password);
```

Declare the features of your task and your histograms

```
HistDB->declareTask("EXAMPLE", "MUON", "GAS", "", true, true, false);

string ServiceName="H1DnodeMF001_EXAMPLE_01/SafetyCheck/Trips";
HistDB->declareHistByServiceName(ServiceName);
ServiceName="H1DnodeMF001_EXAMPLE_01/SafetyCheck/Trips after use of CRack";
HistDB->declareHistByServiceName(ServiceName);
ServiceName="H2DnodeMF001_EXAMPLE_01/OccupancyMap/Hit Map_$$_Region M1R1";
HistDB->declareHistByServiceName(ServiceName);
ServiceName="H2DnodeMF001_EXAMPLE_01/OccupancyMap/Hit Map_$$_Region M1R2";
HistDB->declareHistByServiceName(ServiceName);
ServiceName="H2DnodeMF001_EXAMPLE_01/OccupancyMap/Hit Map_$$_Region M3R1";
HistDB->declareHistByServiceName(ServiceName);
HistDB->commit(); // needed to send histogram buffer to DB
```

Now declare an histogram to be produced at analysis level by some algorithm, and an automatic check to be performed on it (the declaration of algorithms should be done by the developers of analysis library)

```
HistDB->declareCreatorAlgorithm("Subtraction", 2,
                               "bin-by-bin subtraction");
std::vector<OnlineHistogram*> sources;
sources.push_back(HistDB->getHistogram
                  ("EXAMPLE/SafetyCheck/Trips"));
sources.push_back(HistDB->getHistogram
                  ("EXAMPLE/SafetyCheck/Trips after use of CRack"));
OnlineHistogram* htrips=HistDB->declareAnalysisHistogram("Subtraction",
                  "Trips due to CRack", sources);

std::string mypar[1]={"Max"};
HistDB->declareCheckAlgorithm("CheckMax", 1, mypar,
                             "Checks all bins to be smaller than Max");
```

```

if (htrips) {
    std::vector<float> warn(1,100.);
    std::vector<float> alarm(1,500.);
    htrips->declareAnalysis("CheckMax", &warn, &alarm);
}

```

Now create a page and edit the display options of its histograms. Start by setting options for the whole histogram set

```

OnlineHistogram* h1=HistDB->getHistogram
    ("EXAMPLE/OccupancyMap/Hit Map_$$_Region M1R1");
OnlineHistogram* h2=HistDB->getHistogram
    ("EXAMPLE/OccupancyMap/Hit Map_$$_Region M1R2");
OnlineHistogram* h3=HistDB->getHistogram
    ("EXAMPLE/OccupancyMap/Hit Map_$$_Region M3R1");

OnlineHistPage* pg=HistDB->getPage("Example Page","Examples");
pg->declareHistogram(h1,0. ,0. ,0.5,0.5);
pg->declareHistogram(h2,0. ,0.5,0.5,0.5);
pg->declareHistogram(h3,0. ,0. ,0.5,1. );
pg->save(); // // needed to actually send page definition to DB

int lc=2, fs=7, fc=3;
float ymax=20000.;
h1->setHistoSetDisplayOption("LINECOLOR", (void*) &lc);
h1->setHistoSetDisplayOption("FILLSTYLE", (void*) &fs);
h1->setHistoSetDisplayOption("FILLCOLOR", (void*) &fc);
h1->setHistoSetDisplayOption("YMAX", (void*) &ymax);

```

to set more specific display options for single histograms or histograms in a given page, remember to initialize the option set if you want to keep the less specific options

```

lc=3;
h2->initDisplayOptionsFromSet();
h2->setDisplayOption("LINECOLOR", (void*) &lc);

fc=4;
h3->initHistoPageDisplayOptionsFromSet("Example Page");
h3->setHistoPageDisplayOption("FILLCOLOR", (void*) &fc);

```

finally commit changes to the DB

```

HistDB->commit();
delete HistDB;
}

```

6 References

- [1] LHCb Commissioning Group, "Histogramming Framework", EDMS 748834
- [2] "Histogram DB and Analysis Tools for Online Monitoring", EDMS 774740