Histogram DB for Online Monitoring – User's Manual

Issue:	3
Revision:	3
Reference:	
Created:	March 21, 2007
Last modified:	October 26, 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

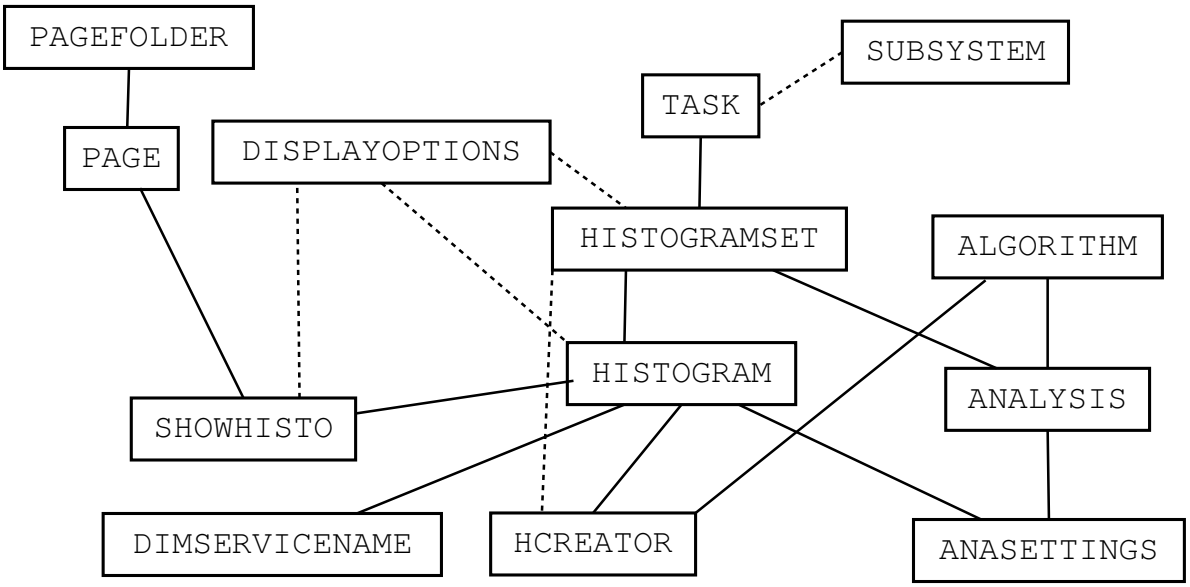

Figure 1 Scheme of DB tables.

1.1 Definition of Histograms

Histograms are uniquely identified by the expression:
Taskname/Algorithmname/HistogramName
The histogram name can contain a subname: *HistogramName = HistogramSetName_$Subname*
Histograms differing only by *Subname* 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

— SaveFrequency
saving frequency for the corresponding saveset

1.1.2 Properties of HISTOGRAMSET

— HSID (integer)

— NHS (integer)
number of histograms in set

— Task (valid TaskName)

— Algorithm (string of max length 100)

— HistogramSetName (string of max length 200)

— Type
'H1D', 'H2D' for normal histograms, 'P1D', 'P2D' for profile histograms, 'CNT' for counters

— 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

— Subname (string of max length 50)

— DIMServiceName (string of max length 130)
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

— ObsoletenessTime (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 are organized in folders (with the syntax of a unix filesystem) and are uniquely identified by their full path name that starting with "/". For example:
/Folder1/page1
/Folder1/SubFolder2/page1
/Folder1/SubFolder2/page2
are all valid page names.

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

— Parent (string of max length 30)
Folder containing this folder (NULL for “root” folders)

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

The unique identifier is the combination (Page,Histo,Instance)

— Page (valid PageName)

— Histo (valid HID)

— Instance (int)
sequence number (starting from 1) to distinguish different instances of the same histogram on the same page

— 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

— HCTYPE
for 'HCREATOR' algorithms, type of histogram generated by this algorithm

— 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)

— HCPARS (arrays(Npars) of floats)
set of needed parameters

2 DB implementation

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

In June 2007 the DB has migrated to the LHCb Oracle server at point8, that can be seen from the cern network only.

The connection string is
<account>/<password>@lbora01:1528/HISTOGRAMDB
where <account> is
HIST_READER (read–only, password="welcome") or
HIST_WRITER

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://www.cern.ch/ggrazian/lhcb/OnlineHistDB/index.php

while the production address (visible only from the CERN network) will be
http://lbweb01:8090/histogramdb

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.

It is also suited for uploading **reference histograms**. Users should provide ROOT files containing all the reference histograms of a given task, that could be simply a saveset files.
These can be uploaded through the "Task Editor" link at
http://lbweb01:8090/histogramdb
A tool is provided to check the file content for missing or unknown histograms. According to the convention of [2], files are saved as
<Reference_root>/<task name>/<datatype>_<startrun>.
where the fields <datatype> and <startrun> allow to define different reference for different run condition and time range.

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 practically any operation on the DB .

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. If you use these methods, 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 to define new DB entries (histograms, tasks, subsystems, pages, algorithms) and query the DB content.
Histograms, tasks and pages can be manipulated using the corresponding classes described in the next sections. For each DB entry, an object can be instantiated through the *getHistogram*, *getTask* and *getPage* methods. Such objects make sense only within the transaction and should never be deleted by the user (they are owned by the OnlineHistDB object and deleted by its destructor).

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

By default, DB errors don't throw exceptions to the client code but for severe inconsistencies (i.e. bugs). The default behaviour can be changed (see section 4.6). Normally, one can detect errors by the method return values and choose if commit or not.
Histogram declarations are a special case since, in order to improve performance, they are stored in a buffer, with default depth 1000, and actually sent to the DB server only when the buffer is full or at commit time. In order to check for errors in histogram declarations, one can use the *sendHistBuffer* method to force buffer sending, and check its return value before committing. Note that if an error occurs and you commit anyway, only all histograms declared before the error are sent to the DB, and the buffer is emptied.

- **OnlineHistDB** (std::string passwd,
  std::string user=OnlineHistDBEnv_constants::ACCOUNT,
  std::string db=OnlineHistDBEnv_constants::DB);

  constructor

- bool **commit**();

  commits all changes to the DB. Returns true if there are no errors.

- 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.

- bool **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,
  float SavingFrequency=0);

  declares a new task to the DB, or updates its configuration

- OnlineHistTask* **getTask**(std::string Name);

  get an OnlineHistTask object, holding informations of an existing task, that can be used to view/edit a task record

- OnlineHistogram* **getHistogram**(std::string Identifier,
  std::string FullPathPageName="_NONE_",
  int Instance = 1);

  gets a pointer to an OnlineHistogram object that can be used to view/edit an histogram record. If FullPathPageName is specified, the default display options for the histogram are those associated to the page (if available). Uses cached histogram objects if available

- OnlineHistogram* **getNewHistogram**(std::string Identifier,
  std::string FullPathPageName="_NONE_",
  int Instance = 1);

  same as getHistogram, but a new object is always created (no caching)

- virtual bool **removeHistogram**(OnlineHistogram* h,
  bool RemoveWholeSet = false);

  removes an histogram, and optionally its full set. (**TEMPORARY METHOD TO BE REMOVED AT PRODUCTION STAGE**)

- OnlineHistPage* **getPage**(std::string Name);

  get an OnlineHistPage object, to create a new page or view/edit an existing one

- bool **removePage**(OnlineHistPage* Page);

  removes completely the page, and all associated options (HANDLE WITH CARE!)

- bool **declareSubSystem**(std::string SubSys);

  declares a subsystem, returning true on success

- 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 **declareHistogram**(std::string TaskName,
  std::string AlgorithmName,
  std::string HistogramName,
  HistType Type);

  declares an Histogram to the DB by its attributes. The enum HistType is defined in class OnlineHistDBEnv

- bool **sendHistBuffer**();

  forces sending histogram declaration buffer to the DB. Returns true if there are no errors.

- OnlineHistogram* **declareAnalysisHistogram**(std::string Algorithm,
  std::string Name,
  std::vector<OnlineHistogram*> &Sources,
  std::vector<float>* Parameters = NULL);

  declares an histogram to be produced at analysis level using algorithm Algorithm. Name is the histogram name. Sources must contain the pointers to the input histograms. Parameters is a pointer to a parameter vector, optionally needed by the algorithm. If the algorithm requires an histogram set as input, use any histogram of the set. Returns the pointer to the new histogram object.

- bool **declareCheckAlgorithm**(std::string Name,
  int Npars,
  std::vector<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.

- bool **declareCreatorAlgorithm**(std::string Name,
  int Ninput=0,
  HistType OutputType = H1D,
  int Npars=0,
  std::vector<std::string> *pars=NULL,
  std::string doc="NONE");

  declares to the DB an available algorithm to produce histograms at analysis time. Ninput is the number of input histograms, Npars the number of optional parameters (pars containing their names), doc is a short description of the algorithm.

- bool **removePageFolder**(std::string Folder);

  removes Page Folder only if it doesn't have pages (useful for cleanup)

- int **getAlgListID**() const ;

  gets the algorithm list version

- bool **setAlgListID**(int algListID);

  sets the algorithm list version (works only for DB admin account)

- int **getAlgorithmNpar**(std::string AlgName,
  int* Ninput = NULL) const;

  gets the number of parameters, and optionally the number of input histograms, needed by algorithm AlgName.

- std::string **getAlgParName**(std::string AlgName,
  int Ipar) const;

  gets the name of parameter Ipar (starting from 1) of algorithm AlgName

- int **getAllHistograms**(std::vector<OnlineHistogram*>* list = NULL,
  std::vector<string>* ids = NULL,
  std::vector<string>* types = NULL);

  gets the full list of histograms. Returns the number of histograms found. Vectors with pointers to OnlineHistogram objects, histogram identifiers, histogram types can optionally created by the user and filled if not null

- int **getHistogramsWithAnalysis**(std::vector<OnlineHistogram*>* list = NULL,
  std::vector<string>* ids = NULL,
  std::vector<string>* types = NULL);

  gets the list of histograms on which some check analysis has to be performed

- int **getAnalysisHistograms**(std::vector<OnlineHistogram*>* list = NULL,
  std::vector<string>* ids = NULL,
  std::vector<string>* types = NULL);

  gets the list of histograms that have to be produced by analysis task

- int **getHistogramsBySubsystem**(std::string SubSys,
  std::vector<OnlineHistogram*>* list = NULL,
  std::vector<string>* ids = NULL,
  std::vector<string>* types = NULL);

  gets the list of histograms related to subsystem SubSys

- int **getHistogramsByTask**(std::string Task,
  std::vector<OnlineHistogram*>* list = NULL,
  std::vector<string>* ids = NULL,
  std::vector<string>* types = NULL);

  gets the list of histograms related to task Task

- int **getHistogramsByPage**(std::string Page,
  std::vector<OnlineHistogram*>* list = NULL,
  std::vector<string>* ids = NULL,
  std::vector<string>* types = NULL);

  gets the list of histograms displayed on page Page

- int **getHistogramsBySet**(std::string SetName,
  std::vector<OnlineHistogram*>* list = NULL,
  std::vector<string>* ids = NULL,
  std::vector<string>* types = NULL);

  gets the list of histograms in a Set

- int **getPageFolderNames**(std::vector<string>& list,
  std::string Parent="_ALL_");

  gets the list of page folders, Parent can be "/", a page folder name or "_ALL_" for all folders

- int **getPageNamesByFolder**(std::string Folder,
  std::vector<string>& list);

  gets the list of pages in a folder

- int **getSubsystems**(std::vector<string>& list);

  gets the list of known subsystems

- int **getTasks**(std::vector<string>& list);

  gets the list of tasks

- int **getAlgorithms**(std::vector<string>& list,
  std::string type="_ALL_");

  gets the list of algorithms, type can be "_ALL_", "CHECK", "HCREATOR"

4.2 OnlineHistogram class

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

- std::string **identifier**() const ;

  full histogram unique identifier Taskname/AlgorithmName/HistogramName

- std::string **page**() const ;

  full path name of the page to which this histogram object is attached

- int **instance**() const ;

  counter (starting from 1) to distinguish several instances of the same histogram on the same page

- std::string **dimServiceName**() const ;

  name of the DIM service that is currently publishing the histogram

- int **hsid**() const ;

  internal histogram set ID

- int **ihs**() const ;

  position of this histogram in set (starting from 1). Can be larger than nhs() if some histogram in the set has been deleted

- std::string **hid**() const ;

  internal histogram ID (equivalent to hsid()/ihs())

- int **nhs**() const ;

  number of histograms in set

- std::string **hstype**() const ;

  histogram type ("H1D", "H2D", "P1D","P2D" or "CNT")

- std::string **hname**() const ;

  histogram name

- std::string **htitle**() const ;

  standard DB histogram title (not necessarly equal to the published ROOT title)

- std::string **hsname**() const ;

  histogram set name

- std::string **subname**() const ;

  subname

- std::string **task**() const ;

  task name

- std::string **algorithm**() const ;

  algorithm name

- std::string **descr**() const ;

  short description of the histogram

- std::string **doc**() const ;

  link to a more extensive documentation

- int **creation**() const ;

  creation date, as a unix timestamp

- int **obsoleteness**() const ;

  if the histogram is no more in use, returns the end–of–validity date as a unix timestamp, otherwise returns 0.

- bool **setPage**(std::string FullPathPageName,
  int Instance=1);

  sets page on which histogram is displayed (reload display options if needed). Histogram has to be already been attached to the page through OnlineHistPage::declareHistogram()

- void **unsetPage**();

  unsets page associated to histogram object

- bool **setDimServiceName**(std::string DimServiceName);

  sets the DIM service name that is currently publishing the histogram. Returns true on success

- bool **setDoc**(std::string doc);

  provide a short description to be optionally printed on the plot

- bool **setDescr**(std::string descr);

  provide a description of the histogram content

- void **dump**();

  dumps histogram data

- void **dumpDisplayOptions**();

  dumps histogram display options

- int **nPageInstances**();

  number of instances of this histogram on any page

- int **nThisPageInstances**();

  number of instances of this histogram on its current page

- typedef enum NONE, SET, HIST, HISTPAGE DisplayOptionMode;

- DisplayOptionMode **domode**() const ;

  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()*

- virtual bool **setDisplayOption**(std::string ParameterName,
  void* value);

  sets a display option. The available parameter names and the corresponding types are listed in section 1.2.1. Returns true on success. This method chooses the most appropriate display option mode calling one of the next methods

- virtual bool **setHistoSetDisplayOption**(std::string ParameterName,
  void* value);

  sets a display option for the whole histogram set

- virtual bool **setHistDisplayOption**(std::string ParameterName,
  void* value);

  sets a display option for the present histogram.

- virtual bool **setHistoPageDisplayOption**(std::string ParameterName,
  void* value,
  std::string FullPathPageName = "_DEFAULT_",
  int Instance=-1);

  sets a display option for the present histogram on page FullPathPageName (default is *page()*)

- virtual bool **unsetDisplayOption**(std::string ParameterName);

  unsets a display option in the current display mode

- virtual bool **unsetHistoSetDisplayOption**(std::string ParameterName);

  unsets a display option for the whole histogram set

- virtual bool **unsetHistDisplayOption**(std::string ParameterName);

  unsets a display option for the present histogram.

- virtual bool **unsetHistoPageDisplayOption**(std::string ParameterName);

  unsets a display option for the present histogram on its page (if defined)

- 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.

- bool **initHistDisplayOptionsFromSet**();

  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 FullPathPageName = "_DEFAULT_",
  int Instance=-1);

  initializes display options associated to this histogram on page FullPathPageName (default is *page()*) with the options defined for the histogram set (if available). Returns true on success.

- bool **initHistoPageDisplayOptionsFromHist**(std::string FullPathPageName = "_DEFAULT_",
  int Instance=-1);

  initializes display options associated to this histogram on page FullPathPageName (default is *page()*) with the options defined for the histogram (if available). Returns true on success.

- bool **getCreationDirections**(std::string &Algorithm,
  std::vector<std::string> &source_list,
  std::vector<float> &parameters);

  for analysis histogram, get the directions for creating histogram

- int **nanalysis**() const ;

  number of analysis to be performed on the histogram set

- void **getAnalyses**(std::vector<int>& anaIDs,
  std::vector<std::string>& anaAlgs) const ;

  get analysy description as vectors of length *nanalysis*() containing the analysis internal IDs and the analysis algorithm names

- boolean **isAnaHist**() const ;

  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.

- bool **setAnalysis**(int AnaID,
  std::vector<float>* warningThr=NULL,
  std::vector<float>* alarmThr=NULL);

  updates parameters for analysis with ID AnaID (for this histogram only). Returns true on success

- bool **getAnaSettings**(int AnaID,
  std::vector<float>* warn,
  std::vector<float>* alarm) const;

  gets parameters for analysis with ID AnaID. Returns true on success

- bool **maskAnalysis**(int AnaID,
  bool Mask=true);

  masks analysis with ID AnaID. Use Mask=false to unmask. Returns true on success

4.3 OnlineRootHist class

This class is intended to be used as an interface between ROOT and the Histogram DB, in order to load or save display parameters trasparently. The user can instantiate objects that can be optionally linked to existing OnlineHistogram and/or TH1 objects.

The identifier must be the DB histogram identifier
Taskname/Algorithmname/HistogramName

- **OnlineRootHist**(std::string Identifier,
  OnlineHistDB *Session = NULL,
  std::string Page="_NONE_",
  int Instance=1);

  constructor using identifier and possibly DB session

- **OnlineRootHist**(OnlineHistogram* oh);

  constructor from existing OnlineHistogram object

- std::string **identifier**() ;

  histogram identifier

- OnlineHistogram* **dbHist**() ;

  corresponding OnlineHistogram object

- TH1* **rootHist**() ;

  corresponding ROOT TH1 object

- virtual bool **setdbHist**(OnlineHistogram* oh);

  link to an existing OnlineHistogram object, returns true on success

- bool **setrootHist**(TH1* rh);

  link to an existing ROOT TH1 object, returns true on success

- bool **connected**() ;

  true if object knows an existing DB session

- OnlineHistDB* **dbSession**() ;

  returns link to DB session

- bool **connectToDB**(OnlineHistDB* Session,
  std::string Page="_NONE_",
  int Instance=1);

  connect to DB session, returns true if the corresponding DB histogram entry is found

- void **setTH1FromDB**();
  updates ROOT TH1 display properties from Histogram DB (via OnlineHistogram object) (normally called when connecting)
- void **setDrawOptionsFromDB**();
  updates current drawing options from Histogram DB (via OnlineHistogram object)
- bool **saveTH1ToDB**();
  saves current ROOT display options to OnlineHistogram object and to Histogram DB
- virtual void **Draw**();
  calls TH1 Draw method, calls setDrawOptions()

4.4 OnlineHistTask class

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

- std::string **name**() ;
  task name
- int **ndet**() ;
  number of associated subdetector/subsystems (up to 3)
- std::string **det**(int i) ;
  name of associated subdetector/subsystems (-1 $<$ i $<$ ndet())
- bool **runsOnPhysics**() ;
  true if task is configured to run for physics events
- bool **runsOnCalib**() ;
  true if task is configured to run for calibration events
- bool **runsOnEmpty**() ;
  true if task is configured to run for empty events
- float **savingFrequency**() ;
  task saving frequency
- std::string **reference**() ;
  location of latest reference file
- bool **setSubDetectors**(std::string SubDet1="NULL",
  std::string SubDet2="NULL",
  std::string SubDet3="NULL");
  sets the associated subdetector/subsystems. "NULL" unsets the value
- bool **addSubDetector**(std::string SubDet);
  adds an associated subdetector/subsystems, returning true on success
- bool **setRunConfig**(bool RunsOnPhysics,
  bool RunsOnCalib,
  bool RunsOnEmpty) ;
  sets run configuration bits
- bool **setSavingFrequency**(float SavingFrequency) ;
  sets task saving frequency
- bool **setReference**(std::string Reference) ;
  sets the location of latest reference file

4.5 OnlineHistPage class

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

The page layout is saved to the DB only through an explicit call of the *save()* method.

- int **nh**() const ;
  number of histograms on page
- const std::string& **name**() const ;
  page name (with full path)
- const std::string& **folder**() const ;
  page folder name
- const std::string& **doc**() const ;
  short page description
- const bool **syncWithDB**() const ;
  check if the page object is in sync with the DB
- bool **setDoc**(std::string Doc) ;
  set short page description
- OnlineHistogram* **declareHistogram**(OnlineHistogram* h,
  float Cx,
  float Cy,
  float Sx,
  float Sy,
  unsigned int instance=1);
  adds or updates an histogram on the page. Use instance > 1 to use the same histogram more than once. Returns the object attached to page (the input one, or a new copy if a new instance needs to be created), or NULL in case of failure.
- OnlineHistogram* **addHistogram**(OnlineHistogram* h,
  float Cx,
  float Cy,
  float Sx,
  float Sy);
  like declareHistogram, but a new instance is always created
- bool **removeHistogram**(OnlineHistogram* h,
  unsigned int instance=1);
  removes histogram from page, or one of its instances, returning true on success
- bool **removeAllHistograms**();
  clean up the page removing all histograms
- void **getHistogramList**(std::vector<OnlineHistogram*> *hlist) ;
  fills the hlist vector with pointers to the histograms attached to this page
- bool **getHistLayout**(OnlineHistogram* h,
  float &Cx,
  float &Cy,
  float &Sx,
  float &Sy,
  unsigned int instance=1) const;
  get the layout of given histogram. returns false if histogram is not found

- bool **save**();
  save the current page layout to the DB

4.6 OnlineHistDBEnv class

All previous classes derivate from this one, that has a few public members:

- typedef enum H1D, H2D, P1D, P2D, CNT, SAM HistType;
- int **debug**() const ;
  get verbosity level (0 for no debug messages, up to 3)
- void **setDebug**(int DebugLevel) ;
  set verbosity level
- int **excLevel**() const ;
  exception level: 0 means never throw exc. to client code, 1 means only severe errors (default value), 2 means throw all exceptions. The default value is 1, meaning that exceptions are thrown only in case of severe DB inconsistency. All other errors, e.g. syntax errors, can be checked from the method return values and from the warning messages on the standard output. exception can catched using *catch(SQLException ex)*
- void **setExcLevel**(int ExceptionLevel) ;
- std::string **PagenameSyntax**(std::string fullname,
  std::string &folder);
  check the syntax of the page full name, returning the correct syntax and the folder name
- void **errorMessage**(std::string Error) const;
  standard way to dump error message

Note that the debug and exception levels defined for the OnlineHistDB object are propagated to all new objects created through the *getPage* and *getHistogram* methods.

5 Code Example

```
#include <OnlineHistDB/OnlineRootHist.h>
#include <OnlineHistDB/OnlineHistDB.h>

int main ()
{
 OnlineHistDB *HistDB = new OnlineHistDB(PASSWORD,
                                         OnlineHistDBEnv_constants::ACCOUNT,
                                         OnlineHistDBEnv_constants::DB);

 bool ok=true;

 ok &= HistDB->declareTask("EXAMPLE","MUON","GAS","",true,true,false);
 OnlineHistTask* mytask = HistDB->getTask("EXAMPLE");
 if (mytask)
   mytask->setSavingFrequency(3.5);

 if (ok) {
   string ServiceName="H1D/nodeMF001_EXAMPLE_01/SafetyCheck/Trips";
   HistDB->declareHistByServiceName(ServiceName);
```

```
  ServiceName="H1D/nodeMF001_EXAMPLE_01/SafetyCheck/Trips_after_use_of_CRack";
  HistDB->declareHistByServiceName(ServiceName);
  ServiceName="H2D/nodeMF001_EXAMPLE_01/OccupancyMap/Hit_Map_$Region_M1R1";
  HistDB->declareHistByServiceName(ServiceName);
  ServiceName="H2D/nodeMF001_EXAMPLE_01/OccupancyMap/Hit_Map_$Region_M1R2";
  HistDB->declareHistByServiceName(ServiceName);
  ServiceName="H2D/nodeMF001_EXAMPLE_01/OccupancyMap/Hit_Map_$Region_M3R1";
  HistDB->declareHistByServiceName(ServiceName);

  HistDB->declareHistogram("EXAMPLE","Timing","Coincidences",OnlineHistDBEnv::H1D);
  HistDB->declareHistogram("EXAMPLE","Timing","Time_of_flight",OnlineHistDBEnv::H1D);

  ok &= HistDB->sendHistBuffer(); // needed to send histogram buffer to DB
}
OnlineHistogram* thisH = HistDB->getHistogram("EXAMPLE/Timing/Time_of_flight");
if(thisH)
  thisH->setDimServiceName("H1D/nodeA01_Adder_01/EXAMPLE/Timing/Time_of_flight");
```

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 normally done by the developers of analysis library)

```
HistDB->declareCreatorAlgorithm("Subtraction",2,OnlineHistDBEnv::H1D,0,NULL,
                                "bin-by-bin subtraction");

OnlineHistogram* s1=HistDB->getHistogram("EXAMPLE/SafetyCheck/Trips");
OnlineHistogram* s2=HistDB->getHistogram(
                     "EXAMPLE/SafetyCheck/Trips_after_use_of_CRack");
OnlineHistogram* htrips=0;
std::vector<OnlineHistogram*> sources;
if(s1 && s2) {
  sources.push_back(s1);
  sources.push_back(s2);
  htrips=HistDB->declareAnalysisHistogram("Subtraction",
                                          "Trips_due_to_CRack",
                                          sources);
}

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

if (htrips && algok) {
  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.

```
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");
```

```
if (h1 && h2 && h3) {
  std::string hcpar[2]={"w1","w2"};
  algok = HistDB->declareCreatorAlgorithm("Weighted mean",
                                          2,
                                          OnlineHistDBEnv::H1D,
                                          2,
                                          hcpar,
                "weighted mean of two histograms with weights w1 and w2");
  sources.clear();
  sources.push_back(h1);
  sources.push_back(h2);
  std::vector<float> weights(2,1.);
  weights[1]=0.5;
  if(algok) HistDB->declareAnalysisHistogram("Weighted mean",
                                             "Silly plot",
                                             sources,
                                             &weights);

  OnlineHistPage* pg=HistDB->getPage("/Examples/My Example Page");
  if(pg) {
    pg->declareHistogram(h1,0. ,0. ,0.5,0.5);
    pg->declareHistogram(h2,0. ,0.5,0.5,0.5);
    pg->declareHistogram(h3,0.5,0.5,0.5,0.4);
    pg->save(); // needed to sync pg object with DB

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

    h1->dump();

    // second instance of h1
    OnlineHistogram* newh = pg->declareHistogram(h1,0.5,0. ,0.5,0.4,2);
    pg->save();
    ymax=200000.;
    newh->setDisplayOption("YMAX",(void*) &ymax);
    newh->unsetDisplayOption("FILLCOLOR");
    lc=4;
    newh->setDisplayOption("LINECOLOR",(void*) &lc);
    newh->dump();

    // link a DB histogram with a ROOT histogram
    TH1F* rh = new TH1F("root histogram",
                        "EXAMPLE/OccupancyMap/Hit_Map Region_M1R1",100,0,1);
    OnlineRootHist* orh=new OnlineRootHist(newh);
    orh->setrootHist(rh);
    // display options are passed to the ROOT object
    cout << "Line color of ROOT histo is  "<< orh->rootHist()->GetLineColor() <<endl;
  }
}
```

Now display the list of pages with their histograms

```
std::vector<string> folders;
```

```
std::vector<string> pages;
std::vector<OnlineHistogram*> histos;
int nfold=HistDB->getPageFolderNames(folders);
int i,j,k;
for (i=0;i<nfold;i++) {
  cout << "Folder " << folders[i] <<endl;
  pages.clear();
  int np=HistDB->getPageNamesByFolder(folders[i],pages);
  for (j=0;j<np;j++) {
    cout << "     Page " << pages[j] <<endl;
    histos.clear();
    int nh=HistDB->getHistogramsByPage(pages[j],&histos);
    for (k=0;k<nh;k++) {
      cout << "           Histogram " << histos[k]->identifier() ;
      if (histos[k]->instance()>1)
        cout << " (Instance "<<histos[k]->instance()<<")";
      cout <<endl;
    }
  }
}
```

and the lists of subsystems, tasks, and algorithms:

```
std::vector<string> mylist;
cout << "----------------------------------------"<<endl;
int nss=HistDB->getSubsystems(mylist);
for (i=0;i<nss;i++) {
  cout << "Subsys "<<mylist[i]<<endl;
}
mylist.clear();
cout << "----------------------------------------"<<endl;
nss=HistDB->getTasks(mylist);
for (i=0;i<nss;i++) {
  cout << "Task "<<mylist[i]<<endl;
}
mylist.clear();
cout << "----------------------------------------"<<endl;
nss=HistDB->getAlgorithms(mylist);
for (i=0;i<nss;i++) {
  cout << "Algorithm "<<mylist[i]<<endl;
}
```

finally commit changes to the DB if there were no errors

```
if (ok)
  HistDB->commit();
else
  cout << "commit aborted because of previous errors" <<endl;

HistDB->setDebug(3); // close transaction verbosely
delete HistDB;
}
```

6 References

[1] LHCb Commissioning Group, “Histogramming Framework”, EDMS 748834

[2] “Histogram DB and Analysys Tools for Online Monitoring”, EDMS 774740