# Parameter initialization in Hydra2 and HGeant2

Ilse Koenig, GSI Darmstadt

May 5, 2017

# Contents


**Contents** iii

**1 Introduction** 1

**2 The runtime database** 3
- 2.1 Schematic overview . . . 3
  - 2.1.1 The building blocks for parameter initialization and output . . . 3
  - 2.1.2 A simple initialization macro . . . 6
  - 2.1.3 Initialization in an analysis macro . . . 8
- 2.2 Parameter containers . . . 8
  - 2.2.1 Base class HParSet . . . 8
  - 2.2.2 Tree-style parameter containers . . . 10
  - 2.2.3 Condition-style parameter containers . . . 11
- 2.3 Parameter container factories . . . 11
- 2.4 Parameter input/output . . . 12
  - 2.4.1 Class Design . . . 12
  - 2.4.2 The Oracle interface HParOra2Io . . . 13
  - 2.4.3 The ROOT file interface HParRootFileIo . . . 14
  - 2.4.4 The ASCII file interface HParAsciiFileIo . . . 15
- 2.5 Creation of parameter files for the analysis . . . 15
  - 2.5.1 Parameter file for a single run . . . 15
  - 2.5.2 Parameter file generated with HParamFileGenerator . . . 16

**3 The HADES geometry** 17
- 3.1 Overview . . . 17
- 3.2 Geometry classes . . . 18
- 3.3 Detector sets . . . 18
- 3.4 HADES geometry formats . . . 21
  - 3.4.1 Volume definition . . . 21
  - 3.4.2 Volume shapes . . . 23
  - 3.4.3 Material and media definition in HGeant2 . . . 26
  - 3.4.4 Hit definition in HGeant2 . . . 27
- 3.5 Geometry interface for HGeant2 and ROOT . . . 27
  - 3.5.1 Configuration file geaini.dat . . . 27
  - 3.5.2 Keywords and file extensions . . . 28
  - 3.5.3 Production of geometry ASCII files for HGeant2 . . . 29
  - 3.5.4 Initialization from ASCII Files . . . 30
  - 3.5.5 Initialization of the ROOT TGeoManager . . . 31
  - 3.5.6 Comparison of geometry versions . . . 32
  - 3.5.7 Storing a new geometry version in Oracle . . . 33
- 3.6 Geometry parameter containers for the analysis . . . 33
  - 3.6.1 The geometry containers for detectors . . . 33
  - 3.6.2 The geometry container SpecGeomPar . . . 34
  - 3.6.3 Alignment . . . 34
  - 3.6.4 Transformations between different coordinate systems . . . 35

**4 The HADES Oracle database** **37**
4.1 Overview . . . 37
4.2 The version management of parameter containers . . . 37
4.3 The WebDB Site . . . 40
4.4 Experiments . . . 40
4.4.1 DAQ runs . . . 41
4.4.2 Simulation projects and reference runs . . . 42
4.5 The WebDB folder Hydra2 Params . . . 43
4.5.1 Parameter releases . . . 44
4.5.2 Overview of parameter containers . . . 44
4.6 Condition-style parameter containers . . . 45
4.6.1 Search for parameter sets . . . 46
4.6.2 Adding a new class . . . 48
4.6.3 Parameter validation . . . 48
4.7 Tree-style parameter containers . . . 49
4.8 Geometry . . . 51
4.9 Explore the WebDB Site . . . 53

# Chapter 1

# Introduction

To run an analysis or a simulation, initialization parameters are needed.

For the **simulation** one needs the full HADES geometry taking into account different target configurations and the different alignment of detectors for a specific beam time. Some detectors need additional parameters stored in a parameter ROOT file, for example ECAL (chapter 3).

Each task in the **analysis** needs special sets of parameters, which are stored in container classes in memory.
Some tasks might share the same container (example: geometry parameters are needed for digitization, tracking, the event display, . . . ).

The parameters are valid for very different time scales.
Once a detector is built, some parameters are fixed for the whole lifetime of this detector (for example number of wires in a given layer of a MDC). Containers holding such data must be initialized only once in an analysis.
Some parameters might change seldom, others more often (for example calibration parameters). In these cases, a re-initialization might be needed during the analysis of several event files (To save memory and time, each container is created only once).
A task might also change parameters during the analysis of an event file and it is then necessary to save these data before a re-initialization.

In the analysis all initialization data are managed by the so-called **runtime database** (chapter 2).

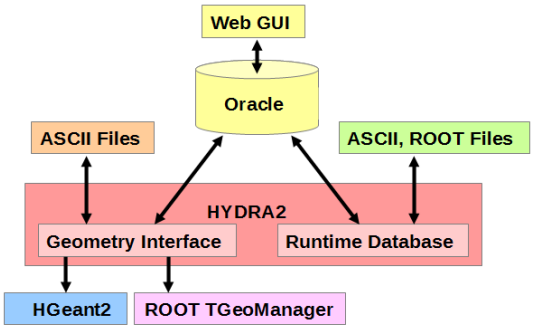



Figure 1.1: Parameter interfaces

All parameters are permanently stored in the **HADES Oracle database** (chapter 4). It stores all actually valid and historic parameters for all beam times and simulations.
Web-based interfaces (**WebDB**) provide additional access to the data in Oracle without running an analysis.

The design of the database tables is not the same for all parameter containers.
Some of the tables are closely related to the technical structure of the detectors, as they hold not only parameters needed by the analysis, but also information about the electronic setup and the cabling. As a result the data are spread over a large number of tables.
Other containers with large amounts of data in a tree like structure are stored in a certain fixed schema (**tree-style parameter containers**).

All these containers need special interface functions to retrieve and store the data.
But most parameter containers developed in the last years use a fixed layout and data are read and stored with a base class interface (**condition-style parameter containers**).

Almost all parameter containers need a **version management** to keep track of changes (see 4.2).
One version is valid for one or more event files characterized by the run id. For the same run, even several versions of parameters might exist, for example a first version of calibration parameters, a second version after a more detailed analysis, etc. It must be possible to read also these historic data in the analysis.

Some parameters even come with different flavors, called **context**, for example wide cuts or strong cuts.

For daily work one needs an additional storage medium for the runtime database: **ROOT files** have been foreseen to store parameter sets temporarily as objects.
For several reasons a local version management has been implemented:

- To run many jobs in parallel on the batch farm.
- To hold the parameters locally to avoid net traffic.
- To distribute the parameters within the collaboration when the direct access to Oracle is not possible or too slow or when the newest data are not yet stored in Oracle.

The ROOT file may contain all parameters to run the analysis of a dedicated list of runs (subset of the parameter versions in Oracle).

To have an easy way to edit or create parameters, an interface to **ASCII files** has been implemented as well. The aim was not to store all data temporarily in ASCII files, but to have an easy way to change parameters with an editor or to compare different versions. A local version management was **not** implemented.

# Chapter 2

# The runtime database

## 2.1 Schematic overview

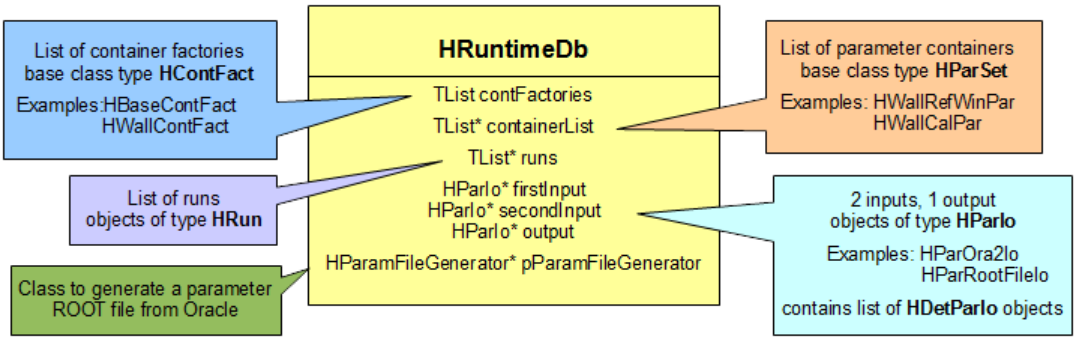



Figure 2.1: HRuntimeDb class diagram

**Features:**
The parameters are stored in parameter containers (see 2.2)
The runtime database allows to create all parameter containers via the container factories (see 2.3).
Once a parameter container is created, it is added in the list of parameter containers and accessible in Hydra2 by all tasks, which need these parameters.
The runtime database allows to specify one or two input sources and takes care, that the parameters are initialized at the beginning of each run with the appropriate version (see 2.4).
The user may specify an output and the runtime database takes care, that the parameters are automatically written to this output before the parameter container is initialized again or before it is deleted.
For each run it stores for each parameter container the input and output versions. This run catalog is, together with the detector setups, written to the ROOT file output to be used as a local database for initialization (see 2.4.3).
For Oracle as input it allows to create a parameter ROOT file for all runs of a specified beam time and time range (see 2.5.2).

### 2.1.1 The building blocks for parameter initialization and output

**Loading the HYDRA libraries**
When the shared libraries are loaded in ROOT, the container factories are created and added to the list of container factories in the runtime database, which is implemented as a singleton.

**Create Hades**
The `Hades` constructor instantiates the runtime database (class `HRuntimeDb`) and creates an object of class `HSpectrometer`.

```
Hades* myHades = new Hades;
HSpectrometer* spec = gHades->getSetup();
HRuntimeDb* rtdb = gHades->getRuntimeDb();
```

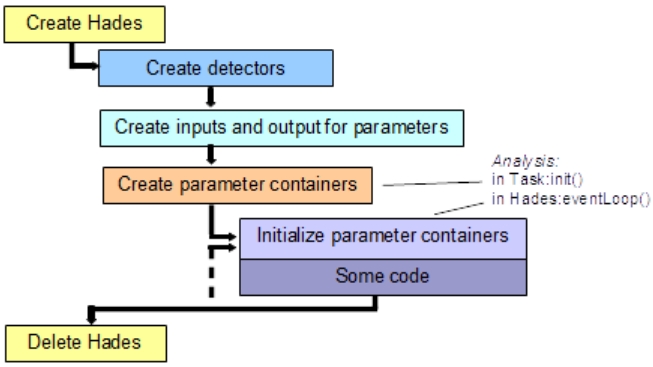



Figure 2.2: Flow chart of initialization

**Create detectors**
In the analysis one must create all detectors with their appropriate setup.
In an initialization macro it is also mandatory for all tree-style parameter containers because they use detector specific I/O classes. Condition-style parameter containers use a base class I/O interface, a detector is not needed.

Each detector has its own class (`HMdcDetector`, `HWallDetector`, ...). They are all derived from the base class `HDetector`. Each detector consists of one or more modules placed in the different sectors or directly in the cave.

The code snippets below creates the Forward Wall detector (with sector = -1, because it sits in the cave) and the MDC detector with 6 sectors, but one module missing (0 = module is absent, 1 = module in setup), and adds them to the list of detectors in the `HSpectrometer` object.

```
HWallDetector* wall=new HWallDetector;
Int_t wallMods[]={1};
wall->setModules(-1,wallMods);
spec->addDetector(wall);

HMdcDetector* mdc=new HMdcDetector;
Int_t mdcMods[6][4]={ {1,1,1,1},
                      {1,1,1,1},
                      {1,1,0,1},
                      {1,1,1,1},
                      {1,1,1,1},
                      {1,1,1,1} };
for(Int_t i=0;i<6;i++) { mdc->setModules(i,mdcMods[i]); }
spec->addDetector(mdc);
```

**Create input and output for parameters**
Here one creates the data sources for the parameters and sets them as first or second input in the runtime database. Additionally one may specify an output.
Possible I/O types are Oracle, ROOT files, ASCII files.

Typically one uses two inputs

- if one wants to merge the data from two input sources into one ROOT file
- if one has some parameters in a file (set this as first input) not stored in Oracle, but all others should be read from Oracle (set as second input)

There is a **simple rule**:
If a parameter container can be initialized from the first input, the second input is not used for this container.

For a more detailed description of the parameter I/O classes and features see 2.4.
Here only a small example, which reads the data from Oracle and writes them to a ROOT file:

```
HParOra2Io* ora=new HParOra2Io;
ora->open();
rtdb->setFirstInput(ora);

HParRootFileIo* output=new HParRootFileIo;
output->open("params.root","RECREATE");
rtdb->setOutput(output);
```

**Create parameter containers**
Typically parameter containers are created by the parameter container factories in the runtime database.
If one wants to use a non-default context for a parameter container, one should set it before the parameter container is created (see 2.3).

```
Bool_t HRuntimeDb::addParamContext(const char context)
```

One creates the containers with

```
HParSet* HRuntimeDb::getContainer(Text_t containerName);
```

The function checks if the container exists already and, if not, creates it. It returns a pointer to the parameter container, but with the base class type, and it must be casted to the concrete type to use the functions not available in the base class, for example:

```
HWallRefWinPar* pPar=(HWallRefWinPar*)(rtdb->getContainer("WallRefWinPar"));
```

One may also create the parameter container via the constructor, but then one must also add it to the runtime database to be later automatically initialized, stored in the output and finally deleted.

```
HWallRefWinPar* pPar=new HWallRefWinPar();
rtdb->addContainer(pPar);
```

In the analysis the parameter containers are created typically in the init functions of the tasks (called in `Hades:init()`).
A few of them are also initialized, for example parameter containers needed to create other parameter containers or to create the data structure for the unpackers. But these are exceptions.

**Initialize parameter containers**
All parameter containers can be initialized with the function call

```
Bool_t HRuntimeDb::initContainers(Int_t runId, Int_t refId, const Text_t fileName);
// runId      number in the event header uniquely defining the hld file or simulation run
// refId      reference run id (default value: -1)
//            if != -1, all containers are initialized with the versions valid for this run
// fileName   name of current event file (default NULL)
```

This function is automatically called in the Hades event loop for the first event of each file.
In a macro it must be called explicitly (eventually several times with different run ids).
The function loops over all parameter containers and calls their init function (see 2.2.1 for details).

In the analysis, the reference run id is set in the data source of the events and always necessary, if parameters are not available for the file to be analyzed. One example are simulation runs where the simulation reference run was **not** set in the configuration file (see 3.5.2 for details).

**Delete Hades**
Before one deletes the Hades object and with it the runtime database, one may want to see its content (in principle one can do this everywhere in the macro, even multiple times)

```
rtdb->saveOutput();
rtdb->print();
```

The function `HRuntimeDb::saveOutput()` loops over all parameter containers and writes them to the output if not yet done. For a ROOT file as output, it also writes the detector setups and the version table, which contains for each run the two input and the output versions, into the output file. Only the output versions are stored in the file. These will be the input versions for the runs when the ROOT file is used as input.

The function `HRuntimeDb::print()` lists the parameter containers, the version table and information about the input(s) and output. (see Example 2.1.2)
**Warning:** If one calls this function without calling `rtdb->saveOutput()` before, the version table might be misleading, because the input and output versions are stored in the version table when the write function is called and for the current run this might have not been done yet.

### 2.1.2 A simple initialization macro

This macro initializes the parameter container HMagnetPar, which contains the magnet current, from Oracle for three runs in beam time APR12 and writes it to a ROOT file.

```
{
  // create the Hades object and get the pointer to the runtime database
  Hades* myHades=new Hades;
  HRuntimeDb* rtdb=gHades->getRuntimeDb();

  // define the input
  HParOra2Io* input = new HParOra2Io;
  input->open();
  rtdb->setFirstInput(input);

  // define the output
  HParRootFileIo* output = new HParRootFileIo;
  output->open("test.root","RECREATE");
  rtdb->setOutput(output);

  // create the parameter container(s)
  HMagnetPar* pMagnetPar = (HMagnetPar*)(rtdb->getContainer("MagnetPar"));

  // initialize the parameter container
  rtdb->initContainers(133602648);  // magnet off
  rtdb->initContainers(133656363);  // magnet on
  rtdb->initContainers(133656526);  // magnet on

  // print the parameters on the screen
  pMagnetPar->printParams();

  // save the output (writes the data to the file)
  rtdb->saveOutput();

  // print content of the runtime database on the screen
  rtdb->print();

  // delete the Hades object
  delete myHades;
}
```

Result of `rtdb->initContainers(133602648);`

```
*********************************************************

   info<HRuntimeDb>:  initialisation for run id 133602648
*********************************************************

*************************************************************
     Oracle history date: 10-JUL-2015 22:13:30
*************************************************************
MagnetPar initialized from Oracle
```

When called the first time the history date, here the last parameter release for beam time APR12, is set. Then the parameter container is initialized.

Result of `rtdb->initContainers(133656363);`

```
   info<HRuntimeDb>: MagnetPar written to ROOT file, version: 1
*********************************************************

   info<HRuntimeDb>:  initialisation for run id 133656363
*********************************************************

MagnetPar initialized from Oracle
```

First the parameter container is written to the output with version number 1, then it is initialized for the next run. The magnet current is different. This leads to a new version.

Result of `rtdb->initContainers(133656526);`

```
   info<HRuntimeDb>: MagnetPar written to ROOT file, version: 2
************************************************************

   info<HRuntimeDb>:  initialisation for run id 133656526
************************************************************
```

The parameter container is written to the output with version number 2, then it is initialized for the next run. Because the start time of this run is in the validity time range of the version read for the previous run, the input version is the same. The Oracle read function is not even called.

Result of `pMagnetPar->printParams();`

```
------------------------------------------
------  MagnetPar  ------
--  Context/Purpose:  MagnetCurrentSetValues
--  Description:      Read from Oracle
            Valid for Run Id 133656363
            Status at 10-JUL-2015 22:13:30
------------------------------------------
current: Int_t   2500
------------------------------------------
```

The call of `rtdb->saveOutput()` does not print anything on the screen because the last version was already written into the file. The write function of the parameter container only stores also for this run the last output version in the version table.

Result of `rtdb->print();`

```
------------------------------------------------------------

   info<HRuntimeDb>: actual containers in runtime database
   info<HRuntimeDb>: MagnetPar          Magnet current
   info<HRuntimeDb>: runs, versions
------------------------------------------------------------------------

   info<HRuntimeDb>: run id
------------------------------------------

   info<HRuntimeDb>: container              1st-input 2nd-input    output
   info<133602648>: run: 133602648
   info<133602648>: MagnetPar              133602648         -1         1
   info<133656363>: run: 133656363
   info<133656363>: MagnetPar              133656363         -1         2
   info<133656526>: run: 133656526
   info<133656526>: MagnetPar              133656363         -1         2
------------------------------------------------------------------------

   info<HRuntimeDb>: input/output
------------------------------------------

   info<HRuntimeDb>: first Input:
Oracle-Database: db-hades    Username: hades_ana
History date: 10-JUL-2015 22:13:30
Hydra2 Oracle interface
detector I/Os:  HCondParIo HSpecParIo
------------------------------------------------------------------------

   info<HRuntimeDb>: second Input: none
------------------------------------------------------------------------

   info<HRuntimeDb>: Output:
HParRootFile**          test.root
 HParRootFile*          test.root
  KEY: HMagnetPar       MagnetPar;2      Magnet current
  KEY: HMagnetPar       MagnetPar;1      Magnet current
  KEY: HRun      133602648;1
  KEY: HRun      133656363;1
  KEY: HRun      133656526;1
detector I/Os:  HCondParIo HSpecParIo
```

Result of `delete myHades;`

```
connection to Oracle closed
```

This ROOT file can be used as input to initialize the parameter containers for the three run ids in the ROOT file. Other runs with the same field settings can only be initialized by using one of these run ids as **reference run id**, for example

```
    Int_t refRunId=133656363;
    rtdb->initContainers(133656688,refRunId); // also magnet on
```

The container will be initialized for run 133656688 with the same data as for 133656363 (version 2).

### 2.1.3 Initialization in an analysis macro

In the following only the parts relevant for initialization are described (**without using the DST library**).
One creates in the macro the detectors, the parameter input(s) and eventually a parameter output.
Then one defines the data source(s) and adds one or more files, eventually with a reference run. The data source opens the first file and reads the run id in the header. If a reference run is defined by filename, it opens also this file and reads its run id.
For HLD files one also adds the unpackers in the HLD source.
Then one defines the tasks respectively task sets.

The parameter containers are created when `Hades::init()` is called:

1. Initializes all containers already created.
2. Calls the init function of all detectors
   For the MDC the parameter containers `HMdcRawStruct` and `MdcGeomStruct` are created and initialized (needed to build other parameter containers and the data structures).
3. Calls the init functions of the data sources
   The init function of a HLD source calls the init function of all unpackers, where the lookup tables are created.
4. Calls the init function of all tasks. Here all other parameter containers are created.

At this point all parameter containers are in the runtime database, but besides a few exceptions, **not yet initialized**. This is done in the **Hades event loop**:

1. Before analyzing the first event of a file all containers are initialized.
2. Then the connection to Oracle is closed.
3. The re-init function of all tasks is called.
   Some tasks calculate private data from the parameter containers and use them in the execute function.
   If several files are analyzed in a chain (or for a ROOT file as input generated from several input files) step 1 - 3 is repeated. The connection to Oracle is automatically re-opened and closed again after initialization.

If one wants to change parameters in a container on-the-fly in the macro without creating a parameter file before one can do this directly before the Hades event loop:

- One gets the pointer to the parameter container using the function
  `HParSet* HRuntimeDB::findContainer(Text_t* containerName).`
  It returns the pointer to an existing container (the constructor is not called).
- One initializes the parameter container (by calling its `init()` function)
- Now one changes the parameter(s).
- The parameter container must be set static to avoid a re-initialization
  (by calling its function `setStatic(Boot_t flag=kTRUE).`

## 2.2 Parameter containers

### 2.2.1 HParSet, the base class for all parameter containers

This class defines the common data elements, some basic functionality and the virtual functions each container class must implement for initialization and output.

**Data Members of HParSet:**

- `TString fName, TString fTitle`
  HParSet is derived from the ROOT class **TNamed** and inherits its data members. The name uniquely identifies the container in the runtime database.

- `TString paramContext`
  This is the name of the parameter context. A parameter context allows to support different flavors of the parameters, which are valid for the same run start[1]. The context defined in the constructor is the default context. Others might be set either via the container factory or explicitly in the macro.

- `Bool_t status`
  By default this flag is kFALSE and the parameters are initialized for each run. Once set kTRUE, the container is skipped in the automatic initialization in the runtime database.
  Member functions:
  ```
  void   setStatic(Bool_t flag=kTRUE);
  Bool_t isStatic();
  ```

- `Bool_t changed`
  By default this flag is kFALSE and set kTRUE after each initialization, which signals, that the container must be written to the output before the data are overwritten by the next initialization or before the container is deleted. The write function then again resets the flag.
  Member functions:
  ```
  void   setChanged(Bool_t flag=kTRUE);
  Bool_t hasChanged()();
  ```

- `Int_t versions[3]`
  It stores in versions[1] the version number from the first input, in versions[2] the one from the second input. By default all versions are -1 (not initialized). If both version numbers are -1 after initialization, the parameter container is not written to the output.
  If one fills the parameter container in the macro and not via the runtime database, versions[1] must be set to 1.
  Member functions:
  ```
  void  setInputVersion(Int_t v=-1,Int_t i=0);
  Int_t getInputVersion(Int_t i);
  ```
  The function `void resetInputVersions()` sets all input versions to -1 and the status flag to kFALSE, but only for a non-static parameter container (status flag kFALSE).

- `Text_t detName[20]`
  This is the name of the detector the container belongs to and set in the constructor of the derived class.
  It is mandatory for all parameter containers which might initialize only a subset of the modules or want to check if all modules in the setup are initialized. Geometry containers need it to retrieve the detector name and setup from Oracle.

- `TString author, TString description`
  These are the author and the description of the parameters in Oracle. They must be filled before the data can be written into Oracle.

**Functions implemented in the base class:**

- `virtual Bool_t init(HParIo*);`
  If the detector name is defined, it first creates an array corresponding to the setup array of this detector.
  Then it checks, if the parameter container was initialized the last time from two inputs and in this case it resets the input versions.
  After this it tries to initialize the parameter container from the first input (if defined) in the runtime database by calling the init function in the derived class (described below). If this fails, because the input cannot provide the data or not all of them, it calls also the init function for the second input (if defined).
  If the parameter container is then still not fully initialized it returns kFALSE, otherwise kTRUE.

[1] Beam and simulation runs may have the same run start, but the parameters might be different. To distinguish between them in Oracle, the identifier of the context id is different for the two run types, but the name is the same.

- `virtual Int_t write();`
  This function gets the output pointer from the runtime database and, if defined, with this calls the function in the derived class (described below).

- `void print();`
  It prints information about the container (context, author, description, versions, status, changed ...).

**Functions implemented in the derived class:**

- `virtual Bool_t init(HParIo*,Int_t*);`
  This function initializes the container from an input using the detector specific interface class of type HDetParIo (see section 2.4 for details).

- `virtual Int_t write(HParIo*);`
  It writes the container to an output using the detector specific interface class of type HDetParIo.
  If one wants to write a parameter container into Oracle, one must use this write function with the pointer to the Oracle output as parameter. One cannot use the base class write function without an argument.

- `void clear();`
  This function resets the parameters to the default values and sets the input/output versions to -1.

### 2.2.2 Tree-style parameter containers

These parameter containers are directly derived from HParSet. Some of them have a tree-like class structure.
Since 2002 we use a standardized table layout in Oracle for a certain group of parameter containers:

- They have a tree-like structure: sector, modules, ..., cells.
- Typically they hold a large amount of data.
- They might be closely related to hardware modules (the data stick to the hardware module, not to the position of this module).
- They need a version management.
- The data elements will probably not change anymore in future code releases.
- The data consistency is checked in Oracle via constraints.

Typical examples are the low-level parameter containers as unpacker lookup tables, thresholds, calibration parameters, . . . .

Although the parameters themselves are individual (they may even be stored in several tables), they use a standard layout for the version management.

- **Advantage:**
  - This allows to write parts of the analysis interface code with cut-and-paste or to reuse code (even without using dynamic SQL for performance reasons).
  - It is possible to use a **generic WebDB GUI** for validation and queries.
  - In the ASCII interfaces one can at least partially use template functions for read/write.
- **Disadvantage:**
  - It needs more coding.
  - The Oracle tables, views, interfaces need to be implemented and later eventually changed by an expert.

### 2.2.3 Condition-style parameter containers

If the list of data members of a tree-like parameter container changes, the "expert" must change the Oracle part. This type is therefore not appropriate for parameter containers as for example conditions (this lead to the name) used in high level analysis, where the code is not stable and may change with every major code release.

To overcome this problem a new type of parameter container has been developed. The data are stored in a generic way in Oracle and the interfaces for Oracle, ROOT and ASCII files are almost completely implemented in base classes.

- **Advantage:**
  - Minimizes code development
  - No need for any new development in Oracle (no expert needed)
  - Properly implemented, it is almost code independent. People can use old **and** new code to read the data from Oracle.
  - Allows to store also large arrays or even ROOT objects in binary format and to read these data very fast.
- **Disadvantage:**
  - All parameters are independent. One cannot put any constraints on the data in Oracle. Consistency checks need to be implemented in the code of the parameter container.

The parameter container must inherit from **HParCond**, derived by itself from HParSet.
Three functions must be implemented:

1. `void clear(void);`
   This function is called automatically before an initialization (new version of parameters).
2. `void putParams(HParamList*);`
   It fills all persistent data members into the list for writing (for Oracle and ASCII files).
3. `Bool_t getParams(HParamList*)`
   It fills all persistent data members from the list after reading. It returns false, when the data of a data member is not in the list.

The Oracle and ASCII read interface fills **all** valid parameters for the parameter container into a list in class **HParamList** (with list elements of type **HParamObj**) used for transfer. The `getParams` function takes from the list only the parameters in the actual used code version calling overloaded functions for the different parameter types.
See class documentation for details.

ALL data are stored in Oracle as binaries (small amount of data in RAW format, larger ones as Binary Large Objects (BLOBS). For ROOT objects (Hydra2 classes derived from TObject) also the streamer info is stored.

**Design considerations:**
Try to avoid the use of ROOT objects in HParamList because they cannot be inspected by the ORACLE WebDB interface and they cannot be stored in an ASCII file. Histograms for example can be serialized into a TArrayD with the HHistConverter before putting them to the list. An example is the parameter container HParticleCandFillerPar.

## 2.3 Parameter container factories

Each library, which contains parameter container classes, must also contain a parameter container factory, derived **HContFact**. It is created when the library is loaded into ROOT and added to the list of container factories in the runtime database.

Each factory contains the list of parameter container names and possible contexts in the corresponding library.
Examples: `HBaseContFact` contains all parameter containers in libBase, `HRichContFact` the ones in libRich.

### Functionality

- They allow to create the parameter containers via the function call:
  `HParSet* getContainer(const char* containerName);`
- For each parameter container they know the possible contexts and they take care for the name of the container when it is created.
  Having different names, they can be read and written from/to the same ROOT file without the need to change the version management.
  Naming convention:
  - If no context was specified before, the parameter container is created with its default context and the name is the standard name.
  - For non-default contexts, the name of the parameter container is concatenated as
    ”standard container name” + ”_” + ”context name”
- Different, but somehow related containers may share the same context.
  Setting this context only once guarantees that all these containers are initialized consistently. For example all geometry containers share the same context ”GeomProduction“.

### Usage

The user typically does not access the container factory directly in the macro, but calls functions in the runtime database:

`Bool_t HRuntimeDb::addParamContext(const char* context)`
This function sets the context of all parameter containers, which accept this context. If a parameter container is created later by the factories it will be created with this context.

`HParSet* HRuntimeDb::getContainer(Text_t* standardContainerName)`
The function loops over all container factories to find the factory responsible for the parameter container with the specified name. The factory then checks, if a special context is set, eventually concatenates the name and then checks, if a container with this name eventually exists already in the runtime database. If not, it creates the container and adds it in the runtime database.
The function returns the pointer to the container, which must be casted to the special container class type. It returns NULL, if the container was not created.

## 2.4 Parameter input/output

### 2.4.1 Class Design

The class design for I/O was developed according to the following requirements:

- Provide interfaces to Oracle, ROOT files and ASCII files
- To avoid dependencies, each detector has its own classes implementing the concrete read and write functions
- The actually used I/O is specified in the macro.
- The interface to Oracle is a shared library not included by the other libraries, except libDst.
  To compile this library one needs the Oracle client, which might not be installed.

The base class **HParIo** defines the virtual functions each type of I/O must implement, for example to close the I/O or to check, if the I/O is open.

The derived classes implement the open function (different arguments for the different types). This class is instantiated in the macro and then set as input or output in the runtime database.
It inherits the data member detParIoList from the base class. This list contains the detector I/O classes (with the read and write functions). When the I/O is opened, these detector I/Os are automatically created for all detectors in the actual setup. This is the reason, why the detectors must be created in the macro **before** an I/O is opened.

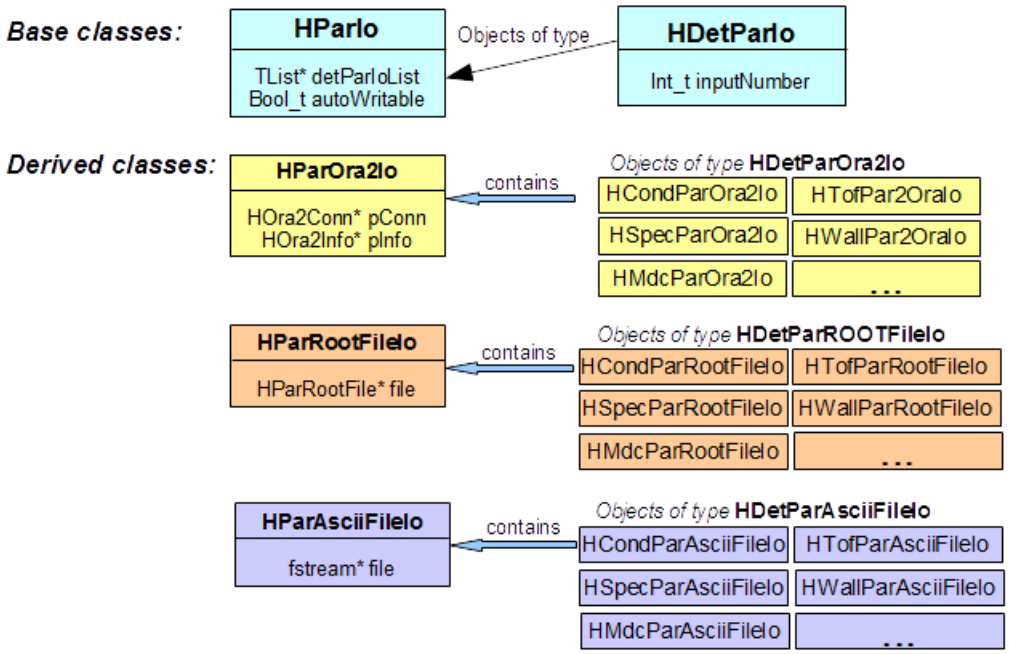



Figure 2.3: Hydra2 I/O classes

Each detector I/O has a name, which is identical for each source type, for example “HCondParIo“. This name is used in the init function for a parameter container to find the corresponding detector I/O. The init function itself does not know, which type of I/O is actually used.

The interfaces ”HCondParIo” and ”HSpecParIo” for condition style parameter containers and parameter containers in libBase are always created in the I/O, even if no detector is created in the macro (for example HCondParOra2Io).

To minimize the code in the specific detector I/O classes, common functionality and utility functions are implemented in the base classes **HDetParRootFileIo**, **HDetParAsciiFileIo** and **HDetParOra2Io**.

### 2.4.2 The Oracle interface HParOra2Io

All interfaces are implemented in the shared library libOra.
To compile this library, one needs the Oracle client. The code uses the Oracle precompiler ProC/C++.
The interface allows to read actual and historic data for all parameter containers and to write parameter containers into Oracle.
Shown here are only typical examples to be used in macros.

The following lines open a connection for the default analysis user HADES_ANA (only privilege to read data and to compile the code) and sets it as first input in the runtime database `rtdb`.

```
HRuntimeDb* rtdb=gHades->getRuntimeDb();
HParOra2Io* ora=new HParOra2Io;
ora->open();
rtdb->setFirstInput(ora);
```

When the function `initContainers()` in the runtime database is called with a certain run id (or reference run id), the parameter containers will be initialized with the data valid for the last parameter release for the corresponding beam time or simulation project. If no parameter release exists yet, the actual valid data for this run will be read.

One may force to read the data valid for this run at a certain point in time by setting the **history date**.
To read always the last version of the data:

```
ora->setHistoryDate("now");
```

To read the data valid in the past (the default for the time is 00:00:00):

```
ora->setHistoryDate(01-JAN-2015 12:00:00");
```

To read the data of a certain parameter release (here parameter release for DST production APR12 generation 7):

```
ora->setHistoryDate("APR12_dst_gen7");
```

If one wants to store data in Oracle one must connect as a user which has the “insert“ privilege for this parameter container. **The interface prompts for the password.**

```
HParOra2Io* ora=new HParOraIo;
Bool_t rc=ora->open("wall_oper");
if (rc) {
   rtdb->setOutput(ora);
   ...
}
```

**HOra2Info** (a data member of HParOraIo) provides various utility functions to retrieve run ids, file names, lists of runs for beam times and simulation projects. Here some examples:

- If one only knows the hld filename one can get the corresponding run id with

  ```
  Int_t runId = ora->getOra2Info()->getRunId("be1211123593401.hld");
  ```

- To get the filename for a run id:

  ```
  TString filename;
  Bool_t found = ora->getOra2Info()->getDaqFilename(134959174,filename);
  ```

- To get the run id for the last run in a specified beam time (useful for testing):

  ```
  Int_t runId = ora->getOra2Info()->getLastRun("APR12");
  ```

- To get a list of HLD files for a specified beam time (mandatory) and (optional) time range and run prefix:

  ```
  TList* listOfFiles = ora->getOra2Info()->getListOfHldFiles("APR12",
                          "20-APR-2012", "20-APR-2012 23:59:59", "be");
  ```

  This list contains objects of type **HRunInfo** with data elements file name, run id, run start, run stop and number of events (>0).

### 2.4.3 The ROOT file interface HParRootFileIo

In the ROOT file parameter containers are stored as objects. The file can hold different versions of the same object (the variable fCycle in TKey).
Every time an object is written it gets automatically a new version incrementing the former version by 1, starting with 1 in each newly created file. The information which run corresponds to which version of each parameter container is stored in the ROOT file together with the data.
By default the `Read()` or `Find()` functions provided by ROOT read the object with the highest version. A retrieval of another version is possible by adding ”`;version number`” to the name of the parameter container.
When initializing from a parameter ROOT file, the first step is to get the version for this run from the version table and then to read this version.
If the run id is not found in the ROOT file, the parameter containers cannot be initialized for this run. But one may use another run in the ROOT file as reference run.

To **write** into a ROOT file, put in the macro (standard ROOT file options RECREATE, NEW, ...):

```
HParRootFileIo* output=new HParRootFileIo;
output->open("wallParams.root","RECREATE");
rtdb->setOutput(output);
```

To **read** from a ROOT file (the default option is `"READ"`):

```
HParRootFileIo* input=new HParRootFileIo;
input->open("wallParams.root");
rtdb->setFirstInput(input);
```

### 2.4.4 The ASCII file interface HParAsciiFileIo

The ASCII interface does not implement a version management.
The reading starts always at the beginning of the file. It reads the first data it finds for the parameter container. A second version in the same file would never be read.

The structure in the parameter file is always the same:

1. The name of the parameter container in [] brackets
2. The data (different formats)
3. At the end a line starting with #

Any comment lines starting with // or # (not in the data block!) are ignored.

To write into an ASCII file, put in the macro:

```
HParAsciiFileIo* output=new HParAsciiFileIo;
output->open("wallParams.txt","out");
rtdb->setOutput(output);
```

To read from an ASCII file (the default option is `"in"`):

```
HParAsciiFileIo* input=new HParAsciiFileIo;
input->open("wallParams.txt");
rtdb->setFirstInput(input);
```

## 2.5 Creation of parameter files for the analysis

To run a full analysis one needs a long list of parameter containers. Some of them have different versions for the runs in a beam time.

**IMPORTANT:** On the batch farm it is **not allowed** to initialize from Oracle more than a few jobs, because it puts quite some load on the Oracle server, if many jobs start in parallel. To avoid overload only a certain number of jobs may read data in parallel (implemented in our software). The others have to wait until the number of jobs reading actively is below a certain limit. This not only slows down your jobs on the batch farm, but hurts also all other people on desktop machines, which want to read parameters from Oracle.
**Use a ROOT parameter file, if you want to run many jobs on the batch farm!**

Furthermore people outside GSI may not have an Oracle client installation and they need a parameter ROOT file to run the analysis locally.

### 2.5.1 Parameter file for a single run

Define in the analysis macro the output in the runtime database. This is typically a ROOT file output, because some parameter container with binary output cannot be written to an ASCII file.

```
HParRootFileIo* output=new HParRootFileIo;
output->open("allParams.root","RECREATE");
rtdb->setOutput(output);
```

Run the analysis of the file with one event.

Then use this file for the analysis of all files which share the same parameters.
**In the macro you must specify the run id, which is in the file, as reference run id.**
This works fine for the analysis of **simulation runs**, where anyhow all files have to be initialized with the run id of the simulation reference run (see 4.4.2 Simulation projects and reference runs). (This is not needed if the reference run id was set in the simulation configuration file (.dat file) and stored this way in the event header.)
For a **high-level analysis** this might work too, because the needed parameter containers typically have only one version for a complete beam time.

Also for testing during code development it is recommended to use such a ROOT file, because typically one analyzes the same run many times.

### 2.5.2 Parameter file generated with HParamFileGenerator

It allows to generate a parameter file from Oracle containing all run ids in a beam time (or some part) with only a few lines of code in the analysis macro:
In an old-fashioned analysis macro:

1. Instead of defining an output in the runtime database, one adds the following lines after the definition of the inputs (rtdb is the pointer to the run time database):

```
// generate a parameter file for beam time AUG14
if (!rtdb->makeParamFile("paramsAug14.root","aug14")) {
  delete gHades;
  return;
}
```

   The ROOT file output is created by the runtime database automatically.
   By specifying a time range one may restrict the beam time for example to a single day or the days where beam was on target.

```
// generate a parameter file for day 239 of beam time AUG14
if (!rtdb->makeParamFile("paramsAug14239.root","aug14",
                         "27-AUG-2014 00:00:00","27-AUG-2014 23:59:59")) {
  delete gHades;
  return;
}
```

2. Add the following line at the end of the macro directly before the delete of Hades:

```
rtdb->saveOutput();
```

**Finally run the macro with 1 or 2 events.**

The function `HRuntimeDb::makeParamFile(...)` takes at least two arguments: the name of the ROOT output parameter file and the experiment name (not case-sensitive).
You may shrink the time range by specifying also a range begin and/or a range end (accepts dates, hld-filenames or run ids).

The ROOT file is opened with option "CREATE" and will fail, if the file exists already. Therefore one should check the return code of the function.

After the first event, the runtime database has the complete list of actually needed parameter containers. The initialization of all runs in the specified range is activated in `HRuntimeDb::saveOutput()`. Using the function HOra2Info::getListOfRuns(...) it gets from Oracle the complete list of runs in the specified range and initializes for each run all parameter containers.
The loop does **not** break, if a parameter container cannot be initialized (this is different from a standard initialization with `HRuntimeDb::initContainers(Int_t)` which would break the loop).

Additionally to the ROOT parameter file a **log-file** is created with the same name, but with extension ".log".
This file contains four parts:

1. The list of runs with filename, run id, start and stop time
2. The list of parameter containers
3. The number of runs, not fully initialized and for these runs the list of missing containers. Search for "**Error**" to find these messages.
4. The version management table for all runs

The function `HRuntimeDb::makeParamFile(...)` may also be called with the name of a **simulation project**, for example "aug14sim". In this case the parameter containers will be initialized for all reference runs in this project.

# Chapter 3

# The HADES geometry

## 3.1 Overview

The full HADES geometry needed to run a **simulation** is stored in Oracle:

- ideal geometry
- different alignment versions for the simulation of beam times
- the information, which detector parts are present in a simulation project (or beam time).

The **geometry interface** (class HGeomInterface) allows

- to read the media, geometry and hit definitions from Oracle and to create the geometry in
  - **HGeant2**
  - the **ROOT geometry modeler** (class TGeoManager) for browsing, drawing, overlap checking and to use eventually Virtual Monte Carlo (VMC) in the future
- to create ASCII files and to read these files for initialization of HGeant2 and the ROOT TGeoManager
- to write the geometry and the parameters for hit definition into Oracle
- to generate geometry files taking into account the alignment of detectors

A **WebDB** interface provides access to the full geometry in **folder Geometry** (see 4.8).

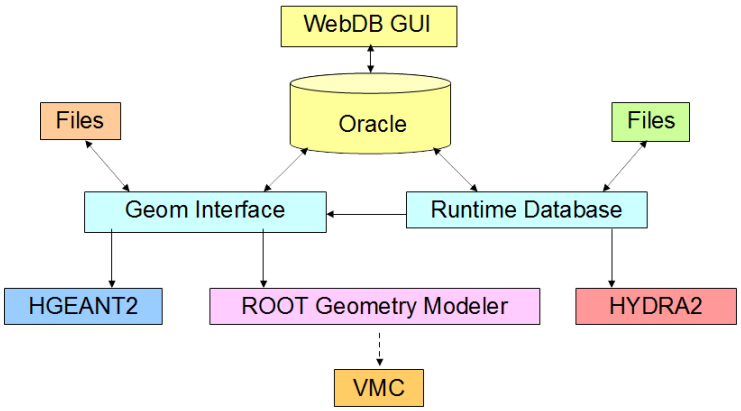



Figure 3.1: Geometry interfaces

In the **analysis** only a small part of the geometry is needed for digitization, tracking, and the event display. It is stored in parameter containers (see Section: 3.6) in the runtime database.

## 3.2 Geometry classes

Fig. 3.2 shows an overview of the geometry classes in Hydra2. The classes in the grey shaded area are implemented in hydra2/base/geometry. They are also used by the geometry parameter containers for the analysis (see 3.6).

All other classes are only needed to create the geometry in HGeant2 or the ROOT TGeoManager (see 3.5) and implemented in hydra2/simulation and hydra2/orasim.

**Hydra2 libraries needed for simulation:**

- libHydra
- libSimulation
- libOraSim (optional, requires the Oracle client installation)

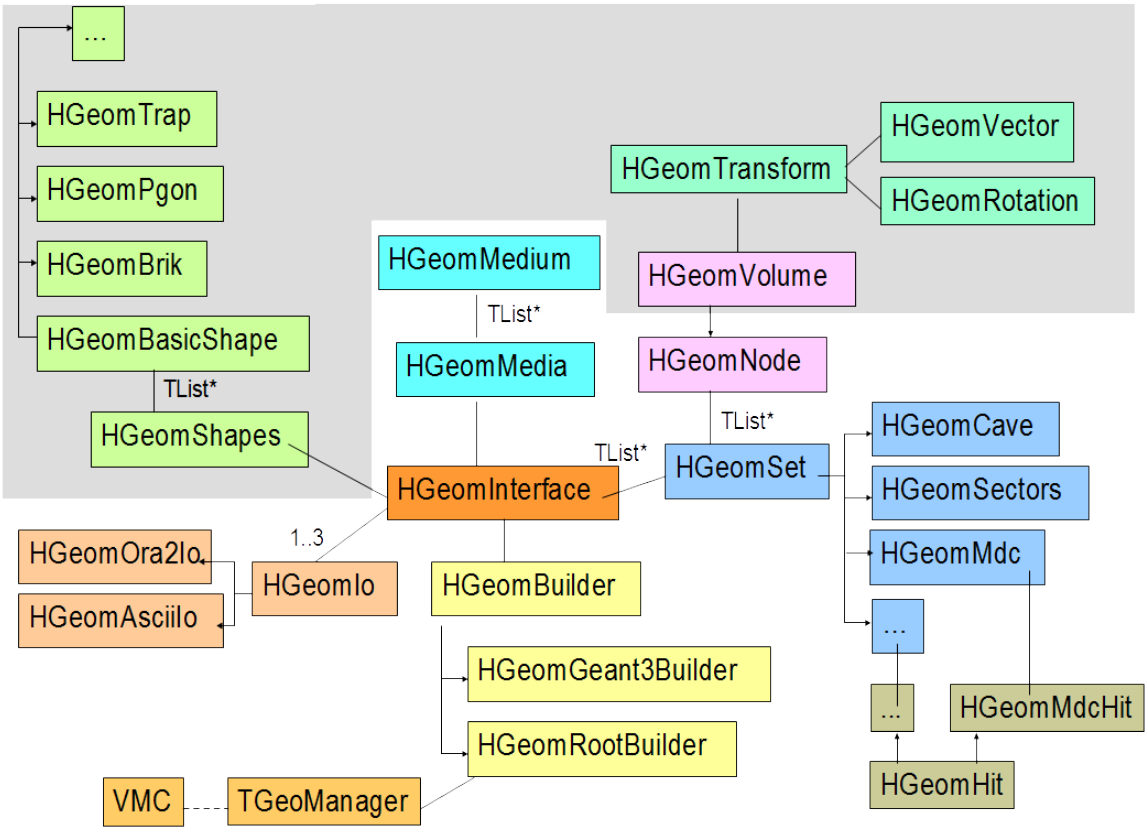



Figure 3.2: Geometry classes

## 3.3 Detector sets

All detector sets are derived from the base class **HGeomSet**, which implements most functionality.
The detectors typically only define some detector specific variables (detector name, number of keep-in volumes and number of modules) and simple functions, which return the name or the substring for keep-in volumes, modules and inner parts.

Detectors with sensitive volumes also have a class for the hit definition, derived from the base class **HGeomHit**.
The detector hit class only implements the function, which returns the IDTYP, the identifier of the hit set (eventually calculated from the name of the sensitive volume).

### Cave

| | |
|---|---|
| *Keyword:* | cave |
| *Geom class:* | HGeomCave |
| *Parameter container:* | HSpecGeomPar |

It is a single volume with name CAVE and has no mother and therefore no transformation.
The origin (0.,0.,0.) of the local coordinate system of the cave is the same as the origin of the field map.
The ASCII file contains only this single volume. All daughters of the cave are defined in separate files.

### Sectors

| | |
|---|---|
| *Keyword:* | sectors |
| *Geom class:* | HGeomSectors |
| *Parameter container:* | HSpecGeomPar |

The set consists of 6 volumes SEC1 ... SEC6 all with the same shape (PGON), medium and dimensions. Looking in beam direction the sectors are counted clockwise starting with the uppermost in positive y-direction.
All components of the translation vector describing the position of the sector coordinate system in the cave must be 0..

### Rich

| | |
|---|---|
| *Keyword:* | rich |
| *Geom class:* | HGeomRich |
| *Parameter container:* | HRichGeometryPar |

The module (volume with name RICH sitting directly in the CAVE) contains a tree of volumes with names all starting with 'R'.
The volume RTAM is the mother volume for the target.

The analysis geometry container HRichGeometryPar is not derived from the base class HDetGeomPar and does not read any information from the standard geometry tables in Oracle. The RICH has its own tables (without version management), which are only for the ideal position (no alignment).

### Target

| | |
|---|---|
| *Keyword:* | target |
| *Geom class:* | HGeomTarget |
| *Parameter container:* | HSpecGeomPar |

The target components (targets and support structures) sit in the RICH volume RTAM. To position the target components properly it is necessary to check the LAB-position of RTAM.
All target components have names starting with 'T'.
The targets must have the name TARG for a single target and TARGxx for a segmented target with identical segments, where xx is the number of the target. Several individual targets (different medium and/or size) must have names starting with 'TX'.
In an alignment geometry version the RICH is shifted, while the target volumes are at ideal z-position in the RICH.

The analysis interface reads only the target volumes from Oracle, not the support structure. To be independent from the RICH and to account for alignment, the transformation of all volumes is stored in the Lab-system.

### MDC

| | |
|---|---|
| *Keyword:* | mdc |
| *Geom class:* | HGeomMdc |
| *Parameter container:* | HMdcGeomPar |

The MDC consists of 4 modules in each sector:

| | |
|---|---|
| plane 1 | DR1M1 ... DR1M6 |
| plane 2 | DR2M1 ... DR2M6 |
| plane 3 | DR3M1 ... DR3M6 |
| plane 4 | DR4M1 ... DR4M6 |

The dimensions of the modules in different sectors are the same, but can have different positions (alignment).
All sub-volumes in the MDC have names starting with ”D1”...”D4”.
The sensitive volumes (layers) must have 4-character names. The 4th character is the number of the layer (for example D1S4 is the sensitive volume in layer 4 of plane 1).

## TOF

| | |
|---|---|
| *Keyword:* | tof |
| *Geom class:* | HGeomTof |
| *Parameter container:* | HTofGeomPar |

The TOF wall consists of 8 TOF modules in each sector:

| *HGeant2 module index* | *Volume names* | *Analysis module index* |
|---|---|---|
| 15 | T15F1 ... T15F6 | 0 (outermost module) |
| 16 | T16F1 ... T16F6 | 1 |
| 17 | T17F1 ... T17F6 | 2 |
| 18 | T18F1 ... T18F6 | 3 |
| 19 | T19F1 ... T19F6 | 4 |
| 20 | T20F1 ... T20F6 | 5 |
| 21 | T21F1 ... T21F6 | 6 |
| 22 | T22F1 ... T22F6 | 7 |

In HGeant the counting is done in positive y-direction and opposite in the analysis. [1]
Each TOF module contains 8 identical sensitive cells (example T15S1 is the TOF rod closest to the beam line).

## RPC

| | |
|---|---|
| *Keyword:* | rpc |
| *Geom class:* | HGeomRpc |
| *Parameter container:* | HRpcGeomPar |

The detector consists of one module in each sector: ”ESKO1” ... ”ESKO6”.
All volumes have names starting with ’E’, the sensitive cells with “EG”.

## SHOWER

| | |
|---|---|
| *Keyword:* | shower |
| *Geom class:* | HGeomShower |
| *Parameter container:* | HShowerGeometry |

The Shower consists of a keep-in volume in each sector with names ”SHK1” ... ”SKH6”.
Each keep-in volume contains 3 modules:

| | |
|---|---|
| module 1 | SH1M1 ... SH1M6 |
| module 2 | SH2M1 ... SH2M6 |
| module 3 | SH3M1 ... SH3M6 |

All daughter volumes in the modules have names starting with ”S1”, ”S2” and ”S3”.
Each module contains only one sensitive volume (for example S1AI in first module SH1M). [2]

## Forward Wall

| | |
|---|---|
| *Keyword:* | wall |
| *Geom class:* | HGeomWall |
| *Parameter container:* | HWallGeomPar |

The Forward Wall consists of a single module with name ”WALL” sitting directly in the CAVE.
All daughter volumes have names starting with ’W’. The name of the sensitive cells start with “W00” followed by a number 1, 2, 3 for the different cell sizes and the copy numbers.

[1] The Tofino, a part of TOF in HGeant1 (module 22...26), was removed in Oracle for HGeant2, but the HGeant2 FORTRAN code was not changed. A simulation with 22 TOF modules (without Tofino) was needed in the past by the kickplane algorithm to create new parameters.

[2] In the analysis the cells are the wire planes S1SW...D3SW, daughters in the center of the sensitive volumes.

### Start

| | |
|---|---|
| *Keyword:* | start |
| *Geom class:* | HGeomStart |
| *Parameter container:* | HStartGeomPar (only Start detector = module 0) |

All volumes start with 'V'. The Start detector module has the name "VSTA" (mother volume "RTAM" in RICH), the Veto module, if present, the name "VVET" (mother volume "RICH"). Some geometry versions contain additional volumes for the delta-electron shielding.
For the pion beam times in 2014, the diamonds in the Start detector (VSTD1 ... VSTD9) were implemented as sensitive volumes with new code in HGeant2 and Hydra2. The Oracle storage and interfaces may change in the future, in case the layout would be different.

### EMC (ECAL)

| | |
|---|---|
| *Keyword:* | emc |
| *Geom class:* | HGeomEmc |
| *Parameter container:* | HEmcGeomPar |

All daughter volumes start with'G'.
The EMC consists of one module in each sector: GMOM1 ... GMOM6, each containing 163 identical daughters. The name of the sensitive volume (Lead glass crystal) is 'GLEA'.
The parameters for the external tracking of the Cherenkov photons are stored in a parameter ROOT file (not in Oracle).

### Coils

| | |
|---|---|
| *Keyword:* | coils |
| *Geom class:* | HGeomCoils |

The magnet coils consist of one module in each sector: CKIV1 ... CKIV6.
All volumes have names starting with 'C'.

### Frames

| | |
|---|---|
| *Keyword:* | frames |
| *Geom class:* | HGeomFrames |

This set contains all support structures and beam pipes, which are not parts of a detector [3], as independent modules. All of them may have substructures.
All volumes have names starting with 'F'. A module can be positioned in the cave ( having a 4-character name "Fxxx") or in the sectors (having 5-character names "Fxxx1" ..."Fxxx6").

### User defined modules

| | |
|---|---|
| *Keyword:* | user |
| *Geom class:* | HGeomUser |

This set allows to read and create up to 9 independent user defined modules (no common keep-in volume) with names "U1KI1" ... "U1KI6" up to "U9KI1" ... "U9KI6". The 5th character is the number of the sector and is missing, if the volume sits directly in the CAVE.
All sub-volumes have names starting with "U1" ... "U9".

## 3.4 HADES geometry formats

### 3.4.1 Volume definition

**GEANT3 and ROOT:**
In **GEANT3 and ROOT** the volume size is specified with the minimum number of parameters needed and for most shapes **relative to the center** of a volume. A TUBE for example is described by three parameters: half length, inner and outer radius. The center of the volume is positioned relative to the center of the mother, described by x, y, z and a $3 \times 2$ rotation matrix.

[3] The MDC frustum and the stiffener bars in MDC4 (do not fit into the keep-in volume) are defined here.

In case the size of the volume changes asymmetrically (for example the tube gets longer at one end, but the daughters should keep their absolute positions), the center of the volume shifts. Not only the transformation of the volume relative to its mother must be changed, but also the transformations of its daughters to keep their position. Furthermore, the orientation of the internal coordinate system is shape dependent, for example different for the trapezoid shapes TRAP and TRD1.
**All sizes, distances and positions are in cm.**

**HADES:**
In the HADES geometry format, each volume has it own local coordinate system, which must **not** be positioned in the center of the volume. The transformation defined by a position vector and a $3 \times 3$ rotation matrix describes the transformation of the local coordinate system of the volume relative to the local coordinate system of its mother.
Some shapes are defined by a list of points (BOX, TRAP, TRD1, ...). This makes it easier to transfer the data from technical drawings into the file format, especially for trapezoids.[4]
The geometry interface calculates automatically the parameters and transformation needed to create the volume inside GEANT or ROOT.
**All sizes, distances and positions are in mm.**

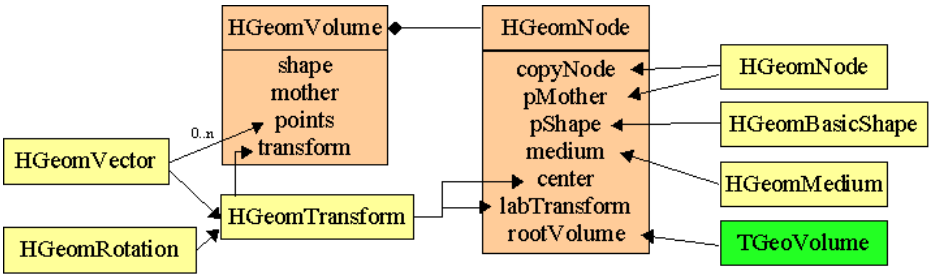



Figure 3.3: The classes HGeomVolume and HGeomNode

A volume (except CAVE, which has no mother and therefore also no transformation) is defined by:

1. name of the volume
2. name of the mother volume
3. GEANT shape of the volume
4. name of the medium (= material)
5. points or parameters from technical drawings (depend on the shape)
6. position (x y z) of the coordinate system, in which the points are given, in the coordinate system of the mother
7. $3 \times 3$ rotation matrix of the coordinate system listed row-wise as vector

**Names of volumes:**
All names start with a detector specific character (see 3.3 Detector sets).
Each volume has a name with at least 4 characters (upper case letters). Only the first 4 characters are used by GEANT (the name of the volume).
All volumes of which several copies exist (several nodes of the same volume) have names with 5 or more characters. These additional characters are always digits typically ranging from 1 to the maximum number of volumes with the same 4-character GEANT name.
For example the outermost TOF (module number 22) contains 8 identical cells T22S1 ... T22S8. The volume T22S is created only once via GSVOLU(...) but positioned 8 times via GSPOS(...).
The names of the sensitive volumes must have a special structure defined for each detector.

[4]Originally it was intended to import the geometry data directly from 3D-CATIA drawings by marking the volume points in CATIA, export them as IGES files and store them in Oracle.

**Name of the mother:**
Each volume is positioned in a mother. The name of the mother has 4 characters (in upper case letters) plus eventually a number, if several copies of the mother exist. This number must be the number of the first node of the mother, typically 1.
(In GEANT only 4 characters are used, but the additional number is needed to find the mother in Oracle or in the file.)

**GEANT Shape:**
Each volume has a shape (4 characters in upper case letters). Implemented in the program are the shapes
BOX , PGON, PCON, TRAP, TRD1, TUBE, TUBS, CONE, CONS, SPHE, ELTU (see section 3.4.2).

**Medium:**
Each volume is filled with a medium given by the name of the medium (see section 3.4.3). If not initialized from Oracle all media must be defined in a common media file.
The predefined media in GEANT are **not** used.

**Points:**
Each volume has parameters describing the dimensions. The number of these parameters and their meaning depend on the shape of the volume. This is explained in section 3.4.2.

**Coordinate system:**
Each volume has a local coordinate system in which the parameters are defined.
Its position and orientation relative to the coordinate system of the mother volume is described by a translation vector T and a 3x3 rotation matrix R according to the equation

```
x=R*x'+T
```

where x is the vector in the coordinate system of the mother and x' the vector in the coordinate system of the daughter. The matrix R is listed row-wise as a vector with 9 components.
Each shape has its own intrinsic orientation, which is defined similar to the GEANT definition except for TRAP and TRD1 (see 3.4.2).
As long as all volumes in a module are not rotated one may use the coordinate system of the module as the base coordinate system in which all points are given.

**Important:** The rotation matrix must be specified with a precision of at least $10^{-6}$. Otherwise one may get a lot of warnings from GEANT that the coordinate system is not orthogonal. The ROOT overlap checker would also detect them as overlaps or extrusions.
Furthermore it may cause rounding problems when storing the geometry in Oracle, because here the transformations of all inner volumes are stored relative to the module coordinate system even for multiple rotations.

Volumes of which several copies exist may use a shortened input structure in the ASCII file:
For the first volume all information above must be defined. For the other copies specified later in the file one must only specify

- name of the volume
- name of the mother
- translation vector of the coordinate system
- rotation matrix of the coordinate system

### 3.4.2 Volume shapes

Each shape is defined by a variable number of 'points' (one in each line of the input file), each having 1-3 components. Apart from the shapes with 8 corners (BOX , TRAP, TRD1) the parameter are partially similar to the ones in GEANT. Each shape has an intrinsic coordinate system. They have the same orientation as in GEANT except for TRAP and TRD1.

**BOX** (class HGeomBrik)
It has 8 corners described by the x, y, z coordinates.
For example the master volume CAVE is implemented as a large BOX.

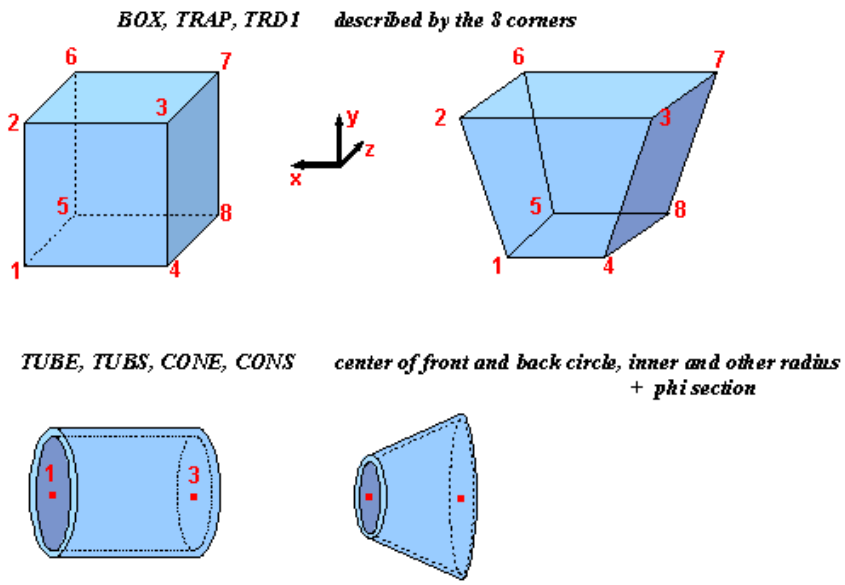



Figure 3.4: Some volume shapes

**TRAP** (class HGeomTrap)
The coplanar front and back planes are trapezoids with centers not necessarily on a line parallel to the z-axis. Typical examples are the MDC modules with back planes larger than the front planes.
A TRAP has 8 corners described by the x, y, z coordinates. The corners are counted clockwise starting at the lower left corner of the front plane.
The intrinsic coordinate system of a TRAP is different from the one in GEANT. The y-axis points from the smaller side in x-direction to the larger one. A TRAP not rotated in a BOX has the same intrinsic coordinate system as the BOX. In GEANT the y- and x-axis point in the opposite directions.

**TRD1** (class HGeomTrd1)
The coplanar front and back planes are trapezoids of the same size and with centers on a line parallel to the z-axis. Examples are the MDC wire planes.
The shape has 8 corners described by the x, y, z coordinates. In HADES the intrinsic coordinate system of a TRD1 is the same as for a TRAP, different from the definition in GEANT.

**PGON** (class HGeomPgon)
The polygon has a variable number of points with 1-3 components:

| | |
|---|---|
| point 0 | NZ number of planes perpendicular to the z-axis where the section is given |
| point 1 | azimuthal angle PHI1 at which the volume begins<br>opening angle DPHI of the volume<br>number NPDV of sides of the cross section between the phi limits |
| point 2ff | z coordinate Z of the section<br>inner radius RMIN at position z<br>outer radius RMAX at position z |

The sectors SEC1...SEC6 are implemented as PGON.

**PCON** (class HGeomPcon)
The polycon has a variable number of points with 1-3 components:

| | |
|---|---|
| point 0 | NZ number of planes perpendicular to the z-axis where the section is given |
| point 1 | azimuthal angle PHI1 at which the volume begins<br>opening angle DPHI of the volume |
| point 2ff | z coordinate Z of the section<br>inner radius RMIN at position z<br>outer radius RMAX at position z |

The RICH volume RMET filled with Methan is implemented as PCON.

**SPHE** (class HGeomSphe)
Is is a spherical shell segment described by 3 points with 2 components:

| | |
|---|---|
| point 0 | inner radius RMIN of the shell<br>outer radius RMAX of the shell |
| point 1 | starting polar angle THE1 of the shell<br>ending polar angle THE2 of the shell |
| point 2 | starting azimuthal angle PHI1 of the shell<br>ending azimuthal angle PHI2 of the shell |

The RICH mirror RMIR is implemented as SPHE.

**TUBE** (class HGeomTube)
This shape has 3 points with 2-3 components:

| | |
|---|---|
| point 0 | x, y, z coordinate of the center of the circle at the beginning of the TUBE |
| point 1 | inner radius RMIN<br>outer radius RMAX |
| point 2 | x, y, z coordinate of the center of the circle at the end of the TUBE |

Typical examples are the solid targets (with RMIN = 0).

**TUBS** (class HGeomTubs)
A TUBS is a segment of a TUBE. It has 4 points with 2-3 components:

| | |
|---|---|
| point 0 | x, y, z coordinate of the center of the circle at the beginning of the TUBS |
| point 1 | inner radius RMIN<br>outer radius RMAX |
| point 2 | x, y, z coordinate of the center of the circle at the end of the TUBS |
| point 3 | starting angle PHI1 of the segment<br>ending angle PHI2 of the segment |

Examples are the 60 degree segments of the Carbon beam pipe FOUC1...FOUC6, one in each sector.

**CONE** (class HGeomCone)
This conical tube has 4 points with 2-3 components:

| | |
|---|---|
| point 0 | x, y, z coordinate of the center of the circle at the beginning of the CONE |
| point 1 | inner radius RMN1 at the beginning of the CONE<br>outer radius RMX1 at the beginning of the CONE |
| point 2 | x, y, z coordinate of the center of the circle at the end of the CONE |
| point 3 | inner radius RMN2 at the end of the CONE<br>outer radius RMX2 at the end of the CONE |

An example is the volume TARG (filled with liquid Hydrogen) of the LH2 target.

**CONS** (class HGeomCons)
A CONS is a segment of a CONE. It has 5 points with 2-3 components:

| | |
|---|---|
| point 0 | x, y, z coordinate of the center of the circle at the beginning of the CONS |
| point 1 | inner radius RMN1 at the beginning of the CONS<br>outer radius RMX1 at the beginning of the CONS |
| point 2 | x, y, z coordinate of the center of the circle at the end of the CONS |
| point 3 | inner radius RMN2 at the end of the CONS<br>outer radius RMX2 at the end of the CONS |
| point 4 | starting angle PHI1 of the segment<br>ending angle PHI2 of the segment |

**ELTU** (class HGeomEltu)
This shape has 3 points with 2-3 components:

| | |
|---|---|
| point 0 | x, y, z coordinate of the center of the ellipsoid at the beginning of the ELTU |
| point 1 | semi-axis P1 along x<br>semi-axis P2 along y |
| point 2 | x, y, z coordinate of the center of the ellipsoid at the end of the ELTU |

**Important:** A TUBE, TUBS, CONE, ELTU cannot be rotated by different x- and y-values of the starting and ending circles or ellipsoids. They have to be identical. A rotation can only be defined by a rotation matrix.

### 3.4.3 Material and media definition in HGeant2

- Each medium is characterized by a name
  HADES naming convention: capital letters, ending with **$**.
- The names of the materials and the media in GEANT are identical.
- For each medium all parameters needed by the GEANT routines GSMATE, GSMIXT, GSTMED and GSCKOV are defined.
- The predefined materials in GEANT are not used.

The following parameters are needed:

| *Parameter* | *Description* |
|---|---|
| int ncomp | number of components in the material<br>ncomp = 1 for a basic material<br>>1 mixture, WMAT contains the proportion by weights<br><1 mixture, WMAT contains the proportion by number of atoms |
| float aw[ncomp] | atomic weights A for the components |
| float an[ncomp] | atomic numbers Z for the components |
| float dens | density DENS in g cm(**-3) |
| float radleng | radiation length RADL (only for a basic material) |
| loat wm[ncomp] | weights WMAT of each component in a mixture (only for a mixture) |
| int sensflag | sensitivity flag ISVOL (1 for sensitive media, 0 for insensitive ones) |
| int fldflag | field flag IFIELD |
| float fld | maximum field value FIELDM in kilogauss |
| float epsil | boundary crossing precision EPSIL |
| int npckov | number of values used to define the optical properties of the medium. |

Comments can only be placed **before** a medium definition in separate lines all starting with `//`.

Examples:

```
GOLDTARGET$
1   196.967   79   19.3   0.33508
0   1   15   0.001
0

// sensitive medium with 2 components specified by number of atoms in the mixture
SCINTILLATOR$
-2   12.011   1.008   6   1   1.03   9   10
1   0   15   0.001
0
```

The variable **npckov** is 0 for all optically inactive media except some special media used in the RICH and in the ECAL to track the Cerenkov photons.
These media have additional parameter arrays:

| | |
|---|---|
| float ppckov[npckov] | photon momentum in eV |
| float absco[npckov] | absorption length in case of dielectric and of absorption probabilities in case of a metal |
| float effic[npckov] | detection efficiency |
| float rindex[npckov] | refraction index for a dielectric, rindex[0]=0 for a metal |

The following parameters used in the GEANT tracking routines are normally not read. The default values are -1 and the real values are automatically calculated by GEANT.

| | |
|---|---|
| float madfld | maximum angular deviation TMAXFD due to the field |
| float maxstep | maximum step permitted STEMAX |
| float maxde | maximum fractional energy loss DEEMAX |
| float minstep | minimum value for step STMIN |

To set the values explicitly one must add the following line in the medium file:

```
AUTONULL
```

After this keyword each medium must specify these 4 parameters in one additional line.

### 3.4.4 Hit definition in HGeant2

The file names for the hit definition must have the suffix **.hit** and contain the detector specific keyword “rich“, ”mdc”, ...(see section 3.3 Detector sets).
The files must contain the following parameters, needed for the GEANT routines GSDET and GSDETH:

| *Line* | *Variable* | *Description* |
|---|---|---|
| 1 | string CHSET | the name of the hit set |
| 2 | int NH | the number of hit components (maximum 30) |
| 3 | string array CHNAMH | NH names of the hit components |
| 4 | int array NBITSH | NH numbers of bits in which to pack the components of the hits |
| 5 | float array ORIG | NH numbers of offsets applied before packing the hit values |
| 6 | float array FACT | NH numbers of scale factors applied before packing the hit values |

Comments can only be placed **before** a detector hit definition as separate lines all starting with `//`.

Example:

```
// Forward Wall hit definition
WLSC
11
   X        Y        Z        TOF      PART     ITRA     ELOS     IDET     PMOM     INWV     TLEN
     30       30       30       30        8       14       30       12       30        4       30
 340.00   340.00   340.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
1.0e+04  1.0e+04  1.0e+04  1.0e+12  1.0e+00  1.0e+00  1.0e+07  1.0e+00  1.0e+07  1.0e+00  1.0e+04
```

## 3.5 Geometry interface for HGeant2 and ROOT

### 3.5.1 Configuration file geaini.dat

The configuration file for HGeant2 consists of three parts:

1. The list of GEANT key words and parameters ended with `END`
2. The geometry configuration and eventually a parameter ROOT file
3. The field map, optional list of event input files, configuration of the ROOT output tree and output file name

Part 1 is read only relevant for a simulation and read by the HGeant2 subroutine `hgeakey.F`.
Part 2 and 3 are read in HGeant2 by the c++ function `HGeantInput::readFileNames()` and stored in various data members depending on special keywords or name/file extensions.

Described here is only the initialization part of the geometry.
In HGeant2 the function `hgeomcreatesetup.cc` (called from `ugeom.F`) instantiates the geometry interface `HGeomInterface`, configures it (sets the builder `HGeomGeant3Builder`, file or Oracle input, parameter file), sets the run id and reads and creates the geometry.
In a pure Hydra2 application, for example the creation of geometry files from Oracle (see 3.5.3), part 2 of the configuration file is read by the geometry interface. Part 1 and 3, if existing, are ignored.

Example: Configuration file for an APR12SIM simulation:

```
!
! GEANT key words
!
PMCF 1
AUTO 1
KINE 0
FMAP 2
FPOL 0.7215
MXST 25000
JVER 2 2 0
BEAM 1. 1. 1230. 0. 0. 0.
SECO 3 1
CKOV 1
LOSS 1
DRAY 1
SWIT 0 0 0 0 0 1 0
TRIG TRIGGER_NUM
ETA  37.30 30.20 21.00 4.75 6.0 7.1e-2
```

```
TIME 0 1000000 1000000
SPLIT 1800000000
FILE 1
END              ! End of gffread
//
// ****************************************************************
//
// ---- geometry initialized from Oracle ----------------------------
SimulRefRunDb:    apr12sim_mediumfieldalign_auau
HistoryDateDb:    now
//
// ****************************************************************
//
// ---- field map ---------------------------------------------------
/cvmfs/hades.gsi.de/param/field/fldrpz_unf.map
//
// ---- branches in the ROOT output tree ----------------------------
kine.tup
rich.tup
mdc.tup
tof.tup
shower.tup
wall.tup
rpc.tup
//
// ---- input file(s) -----------------------------------------------
/misc/kempter/projects/AuAu/run_urqmd/out/Au15Au_urqmd_bmax4_1000evts.evt
//
// ----  output file -----------------------------------------------
test.root
```

### 3.5.2 Keywords and file extensions

Special **keywords** (specified as ”keyword” + ”: ” + ”value”) and **file extensions** are used to read and create the geometry.

- **DbSupport:**
  This keyword accepts two values:
  - **ON** (default)
    This creates the Oracle interface class HGeomOra2Io.
  - **OFF** In this case the geometry must be fully initialized from .geo and .hit files. This is the **standard value when running a simulation on the batch farm**.
- **SimulRefRunDb:**
  This is the keyword to read the **full** geometry including the hit definition from Oracle.
  The value is the name of a simulation reference run. This defines the detector setup and the geometry version of each detector part. The corresponding run id is set in the event header of the ROOT output file.

  ```
  SimulRefRunDb:    apr12sim_mediumfieldalign_auau
  ```

- **HistoryDateDb:**
  This is the keyword to set the history date of the geometry in Oracle.
  As in the analysis by default the geometry of the last parameter release for the simulation project is read from Oracle or “ now” if no release exists.

  ```
  HistoryDateDb:    now
  //HistoryDateDb:    01-NOV-2013 00:00:00
  ```

- **ParameterFile:**
  With this keyword one may specify a parameter ROOT file for example needed to create the MDC geometry with wires or to read the histograms for the external tracking of Cerenkov photons in the ECAL modules and the lookup table for the photomultipliers.
- **SimulRefRunId:** (HGeant2 only)
  With DbSupport OFF the run id is by default 0. In the analysis one needs to set the reference run id for initialization.
  As an alternative one can set the run id here and this way add it in the event header of the output file.

- **DebugFile:** (HGeant2 only)
  For debugging of a new geometry it might be helpful to get the parameters calculated by the program and used to call the GEANT FORTRAN routines. The file extension must be ".txt".

- **.geo**
  This is the file extension of the geometry or media ASCII files. For a detector with sensitive volumes also the corresponding .hit file must be defined.

- **.hit**
  This is the file extension of an ASCII file containing the hit definition of a detector with sensitive volumes.

- **_gdb**
  It is possible to mix input from Oracle and input from file. The part of the HADES detector read from Oracle is specified by

```
    "keyword for the detector part" + "_gdb"
```

  **All media must be read from file.**
  In the example below the media and the target are read from file, the CAVE and the RICH from Oracle. All other parts are missing and therefore not created.[5]

```
SimulRefRunDb:    apr12sim_mediumfieldalign_auau
HistoryDateDb:    now
//
/misc/ilse/svn/hydra2/macros/Geom/pion/media_new.geo
cave_gdb
rich_gdb
/misc/ilse/svn/hydra2/macros/Geom/pion/target_new.geo
```

- **.setup**
  Independent if one reads from Oracle or from files, always the full setup of the detectors is read and would be created.
  **In Hydra2 the default setup for the MDC are 24 modules and for the TOF the 8 outer modules.**
  To create only a subset of detectors one must specify a file with extension **.setup** in the configuration file. Below is an example how to define subsets:

```
// ------------------------------------------------------------------------
// Line 1 containes the detector name (lower-case) in [] brackets.
// Line 1 - 6 specifiy the setup in each sector with 1 = "in setup",
//                                                   0 = "not in setup".
// ------------------------------------------------------------------------
// NOV02 MDC setup
[mdc]
SEC1 1 1 1 1
SEC2 1 1 0 0
SEC3 1 1 1 0
SEC4 1 1 1 1
SEC5 1 1 0 0
SEC6 1 1 1 0
```

### 3.5.3 Production of geometry ASCII files for HGeant2

The interface allows to store the geometry read from Oracle in ASCII files. Then these file may be used for initialization for example on the batch farm.

**1. Step: Configuration file**
The file must contain the name of the simulation reference run and eventually a history date, but no .geo or .hit files.
Example: ora2files.dat

```
SimulRefRunDb:    apr12sim_mediumfieldalign_auau
HistoryDateDb:    now
```

[5] By reading only the media from file and some other parts from Oracle it is possible to create only a subset of the HADES detector.

**2. Step: ROOT macro ora2files.C**
One must only specify the configuration file and the output directory.

```
{
  // configuration file
  TString configFile = "ora2files.dat";

  // output directory (must exist)
  TString outputDir = "./geom12001";

  // ******************************************************

  HGeomInterface* interface=new HGeomInterface;

  HGeomOra2Io* oraInput=new HGeomOra2Io;
  oraInput->open();
  interface->setOracleInput(oraInput);

  Bool_t rc=interface->readGeomConfig(configFile.Data());

  HGeomAsciiIo* output=new HGeomAsciiIo;
  output->setDirectory(outputDir.Data());
  interface->setOutput(output);

  Bool_t rc=interface->readAll();
  if (rc) {
    interface->writeAll();
    // HGeomSet* set=interface->findSet("target");
    // if (set) interface->writeSet(set);
    // interface->writeMedia();
  }
  delete interface;
}
```

It generates separate files for the media and all detector parts with file names

```
    "keyword for the detector part" + "actual date and time" + ".geo"
```

and for detectors with sensitive volumes also the hit files with extension ".hit".

The commented lines show an example how to write only a single .geo file.

### 3.5.4 Initialization from ASCII Files

With all .geo and .hit files created from Oracle one can switch OFF the Oracle support and replace the geometry part in the configuation file:

```
//
// SimulRefRunDb:    apr12sim_mediumfieldalign_auau
// HistoryDateDb:    now
//
DbSupport:       OFF
SimulRefRunId:  12001
//
geom12001/media_230615155440.geo
geom12001/cave_230615155440.geo
geom12001/rich_230615155440.geo
geom12001/target_230615155440.geo
geom12001/sect_230615155440.geo
geom12001/mdc_230615155440.geo
geom12001/coils_230615155440.geo
geom12001/tof_230615155440.geo
geom12001/shower_230615155440.geo
geom12001/frames_230615155440.geo
geom12001/start_230615155440.geo
geom12001/wall_230615155440.geo
geom12001/rpc_230615155440.geo
//
geom12001/rich_230615155440.hit
geom12001/mdc_230615155440.hit
geom12001/tof_230615155440.hit
geom12001/shower_230615155440.hit
geom12001/start_230615155440.hit
geom12001/wall_230615155440.hit
geom12001/rpc_230615155440.hit
```

If a file is missing, the corresponding detector part will not be created. This way one can create only a subset of the HADES detector.

### 3.5.5 Initialization of the ROOT TGeoManager

The interface allows to launch the geometry inside ROOT (class TGeoManager) with a special builder class `HGeomRootBuilder` which creates the ROOT objects of type TGeoVolume, TGeoNode, ... with the ROOT specific shape, transformation and material classes.

With the TGeoManager one may

- check a new geometry for overlaps and extrusion
- browse the geometry tree, materials, overlaps and draw volumes and its daughters in 3D as wire frames
- draw the geometry with OpenGL for nice colored pictures

Below an example macro. The configuration file is typically the same as the one for a HGeant2 simulation.

```
{
  // specify configuration file
  TString configFile="geaini.dat";

  HGeomInterface* interface=new HGeomInterface;

  Bool_t rc=interface->readGeomConfig(configFile.Data());
  if (!rc) printf("Read of GEANT config file failed!\n");

  if (rc) rc=interface->readAll();
  if (!rc) printf("Read of geometry failed!\n");

  // create ROOT TGeoManager
  TGeoManager* geom = new TGeoManager("HadesGeom", "HADES geometry");

  // create builder class for ROOT volumes and nodes
  HGeomRootBuilder* builder=new HGeomRootBuilder("builder","geom builder");
  builder->setGeoManager(geom);
  interface->setGeomBuilder(builder);

  if (rc) rc=interface->createAll();
  if (!rc) printf("Creation of geometry failed!\n");

  delete interface;

  if (rc) {
    // check for overlaps and extrusions
    geom->CheckOverlaps(0.0001);  // ROOT uses cm!
    geom->PrintOverlaps();

    // draw geometry and overlaps with TBrowser
    TBrowser* browser=new TBrowser;

    // draw volumes with OpenGL
    TGLViewer* v=(TGLViewer*)gPad->GetViewer3D();
    v->SetStyle(TGLRnrCtx::kWireFrame);
    geom->SetVisOption(0);
    geom->SetMaxVisNodes(4000);
    geom->DefaultColors();
    // draw mother volume of target and Start detector
    TGeoVolume* vol=geom->GetVolume("RTAM");
    vol->Draw("ogl");
    geom->SetVisLevel(1);  // visibility level
  } else {
    delete geom;
  }
}
```

### 3.5.6 Comparison of geometry versions

In Hydra2 it is possible to compare different geometry versions (still missing in the public Oracle WebDB GUI) by using the function `HGeomSet::compare(HGeomSet&)`. It loops over all volumes in the tree, tries to find the volume with the same name in the referenced list and checks it for differences.

Recipe:

1. Create first interface and read the geometry from the first input
2. Create second interface and read the geometry from the second input
3. Find the sets in the first and second interface and compare them

Both inputs might be Oracle or geo files.
This way one can compare two Oracle versions (different simulation reference runs or different history dates for the same reference run), geo files with the geometry in Oracle or different geo file versions.

The macro below compares for example a full geometry read from Oracle with the full geometry read from geo files.

```
{
  // specify the two configuration files
  TString configFile1="apr12sim_from_oracle.dat"; // Oracle
  TString configFile2="apr12sim_from_files.dat";  // geo files

  // read full geometry from first input, here from Oracle
  HGeomInterface* interface1=new HGeomInterface;
  HGeomOra2Io* oraInput=new HGeomOra2Io;
  oraInput->open();
  interface1->setOracleInput(oraInput);
  interface1->readGeomConfig(configFile1.Data());
  interface1->readAll();

  // read full geometry from second input, here from ASCII files
  HGeomInterface* interface2=new HGeomInterface;
  interface2->readGeomConfig(configFile2.Data());
  interface2->readAll();

  // list of all possible detector parts
  TString sets[12] = {"cave", "sect", "rich", "target", "start", "mdc",
                      "tof", "rpc", "shower", "wall", "coils", "frames"};

  HGeomSet *set1 = NULL, *set2 = NULL;

  for (Int_t i=0; i<12; i++) {
    const char* name = sets[i].Data();
    set1=interface1->findSet(name);
    set2=interface2->findSet(name);
    if (set1 == NULL && set2 == NULL) continue;
    if (set1 != NULL && set2 != NULL) {
      cout << "*****************************************************************" << endl;
      cout << "***  "<< name << endl;
      cout << "*****************************************************************" << endl;
      set1->compare(*set2);
    } else {
      cout << "*****************************************************************" << endl;
      if (set1 != NULL && set2 == NULL) cout << name << " missing in config 2" << endl;
      if (set1 == NULL && set2 != NULL) cout << name << " missing in config 1" << endl;
      cout << "*****************************************************************" << endl;
    }
  }

  delete interface1;
  delete interface2;
}
```

The output lists for each detector separately the differences for the name of the mother, name of the medium, the shape, the shape parameters, the positioning vector of the intrinsic coordinate system and the rotation matrix:

`0 = same, 1 = different, same if identical.`

A summary at the end shows if both versions contain the same list of volumes.

Here an example where the medium was changed:

```
...
*************************************************************
***   mdc
*************************************************************
name      mother medium shape   points    pos     rot
-------------------------------------------------------
DR1M1     same
D1F1      same
D1I1      same
D1F2      same
D1I2      same
D1A1      same
D1G1         0       1       0       0       0       0
D1C2         0       1       0       0       0       0
D1A2      same
...
Number of volumes in first list:              169
Number of different volumes:                  94
Number of volumes not found in second list:   0
Number of additional volumes in second list:  0
*************************************************************
...
```

### 3.5.7 Storing a new geometry version in Oracle

The way how to store new geometry versions in Oracle is described in the separate document "**Administration guide for the HADES Oracle database DB-HADES**", chapter "**HGEOM, the geometry account**".
It includes an example macro and screen shots of the WebDB interface to create a new geometry version or to fix a bug, to insert new media and to validate geometry versions for a beam time or simulation project.

## 3.6 Geometry parameter containers for the analysis

### 3.6.1 The geometry containers for detectors

All detectors except RICH [6] have geometry containers derived from the base class `HDetGeomPar`.
In two arrays the parameter container stores the sizes and positions of volumes forming a geometry tree with 2 levels: detector modules and components (the detector cells) in these modules.

1. The array **modules** stores the LAB-transformation of all modules in the setup. Each object of type `HModGeomPar` has a name, a name of the reference module and a pointer to this reference module set during filling. Typically modules of the same type (for example MDC plane 1 modules) are copies in all sectors. Their geometry can be described by one reference module, only the position of the individual modules is different.

2. The second array **refVolumes** stores the geometry volumes of the reference modules (objects of type `HGeomCompositeVolume` derived from base class `HGeomVolume`). Each module holds an array of its components (also of type `HGeomVolume`).
   Each volume has a name, a shape, shape dependent points, a mother name and a transformation relative to the mother (see fig. 3.3).
   **Here the mother of all reference modules is the CAVE.** Its transformation is the LAB-transformation, not the sector transformation.

The Oracle interface reads the geometry of the detector modules and cells from the same tables, which contain the full geometry for simulation. First it reads the names of the modules and cells from tables in the detector accounts.
*As example the MDC geometry container HMdcGeomPar:*
`HMdcParOra2Io::readModGeomNames(...)` reads the module name for each sector and MDC plane,
`HMdcParOra2Io::readLayerGeomNames(...)` reads the name of the sensitive volumes for each MDC plane and layer.

[6] The RICH geometry container RichGeometryParameters (class HRichGeometryPar) is a condition-style parameter container and contains no volume information.

With these names it retrieves the volume parameters using functions in the base class `HDetParOra2Io`. [7]

For **beam runs** typically only the ideal geometry of the detectors and the target is validated, often the same for several beam times. Only the alignment, stored separately in Oracle, changes more often, sometimes even for different DST generations [8].
The analysis interface first reads the (ideal) geometry valid for the run and fills the parameter container.
In a second step it reads the valid alignment data, if existing, and with these LAB-transformations overwrites the transformations in the array **modules** (but not the transformation of the reference module).

For **simulation runs** it only reads the version valid for the run.

### 3.6.2 The geometry container SpecGeomPar

It stores the geometry volumes of the cave, the sectors and the targets (without the target holder and the supporting foils).

| | |
|---|---|
| `HGeomVolume* getCave(void);` | returns the cave |
| `HGeomVolume* getSector(const Int_t n);` | returns the sector with index n (0...5) |
| `Int_t getNumTargets(void);` | returns the number of targets |
| `HGeomVolume* getTarget(const Int_t n);` | returns the sector with index n |

The cave has no mother and no transformation. The transformations of the sectors and the targets are the LAB-transformations (mother CAVE).
The Oracle interface is implemented in `HSpecParOra2Io`. For real runs the targets may have an alignment which overwrites the ideal positions. It is stored in Oracle via the write function.

### 3.6.3 Alignment

In Oracle the alignment is implemented as a standard tree-style parameter container. The write function of the geometry container creates a new version and stores the LAB-transformations of each module, respectively target (see example macro). The version must be validated with the WebDB interface for tree-style parameter containers.

#### Example macro to store the alignment

This macro initializes the geometry parameter container of the Start detector (only module with index 0) from ASCII file for the beam time AUG14 and stores the alignment in Oracle.

```
{
  Hades* myHades=new Hades;
  HRuntimeDb* rtdb=gHades->getRuntimeDb();
  HSpectrometer* spec=gHades->getSetup();

  Int_t startMods[] = {1,0,0,0,0,0,0,0,0,0};
  spec->addDetector(new HStart2Detector);
  spec->getDetector("Start")->setModules(-1,startMods);

  HParAsciiFileIo* input=new HParAsciiFileIo;
  input->open("startGeomParAug14.txt");
  rtdb->setFirstInput(input);

  HParOra2Io* ora=new HParOra2Io();
  ora->open("db-hades","hanal2");
  rtdb->setOutput(ora);

  HStart2GeomPar* pStartGeom = (HStart2GeomPar*)(rtdb->getContainer("Start2GeomPar"));
  rtdb->initContainers(207913886);

  pStartGeom->setAuthor("Ilse Koenig");
  pStartGeom->setDescription(
              "Start detector alignment for beam time AUG14, based on ideal version 6");
  pStartGeom->write(ora);

  delete myHades;
}
```

[7] The ASCII interface is implemented in the base class HDetParAsciiFileIo, the ROOT file interface in HDetParRootFileIo.
[8] Only for the "final" alignment a new full geometry version is produced and stored in Oracle for simulations.

### 3.6.4 Transformations between different coordinate systems

The class HGeomTransform provides functions to transform a point implemented as `HGeomVector` defined in the local coordinate system into the coordinate system of the mother or the daughter.

`HGeomVector HGeomTransform::transFrom(const HGeomVector& p) const`
transforms a point v1 from the local coordinate system to a point v2 in the mother coordinate system.

```
HGeomVector v2=mother.transFrom(v1)
```

`HGeomVector HGeomTransform::transTo(const HGeomVector& p) const`
transforms a point v1 from the local mother coordinate system to a point v2 in the daughter coordinate system.

```
HGeomVector v2=daughter.transTo(v1)
```

The simple macro below with hard-coded numbers for testing shows an example how to transform a MDC Geant hit from the layer coordinate system to a hit in the sector coordinate system.
It needs the initialized (here from Oracle) parameter containers ”SpecGeomPar“ and ”MdcGeomPar“.

```
{
  Hades* myHades=new Hades;
  HSpectrometer* spec=gHades->getSetup();
  HRuntimeDb* rtdb=gHades->getRuntimeDb();

  HMdcDetector* mdc=new HMdcDetector;
  Int_t mdcMods[6][4]=
      { {1,1,1,1},
        {1,1,1,1},
        {1,1,1,1},
        {1,1,1,1},
        {1,1,1,1},
        {1,1,1,1} };
  for(Int_t i=0;i<6;i++) mdc->setModules(i,mdcMods[i]);
  spec->addDetector(mdc);

  HParOra2Io* input = new HParOra2Io;
  input->open();
  rtdb->setFirstInput(input);

  HSpecGeomPar* specGeomPar = (HSpecGeomPar*)(rtdb->getContainer("SpecGeomPar"));
  HMdcGeomPar*  mdcGeomPar  = (HMdcGeomPar*)(rtdb->getContainer("MdcGeomPar"));

  rtdb->initContainers(12000);  // ideal geometry for APR12

  // ------------------------------------------------------------------------------
  // example Geant hit with x, y position of track entering the volume in forward direction
  // ------------------------------------------------------------------------------
  //  HGeantMdc* fGeant;
  //  Int_t sectorNum = (Int_t)(fGeant->getSector());
  //  Int_t moduleNum = (Int_t)(fGeant->getModule());
  //  Int_t layerNum  = (Int_t)(fGeant->getLayer());
  //  Float_t x, y, tof, ptot;
  //  fGeant->getHit(x, y, tof, ptot);

  // Example values for testing:
  Int_t sectorNum = 1, moduleNum = 0, layerNum = 3;
  Float_t x = 0.F, y = 0.F;

  // get sector transformation
  HGeomVolume* sector = specGeomPar->getSector(sectorNum);
  HGeomTransform& sectorTrans = sector->getTransform();

  // get module transformation
  HModGeomPar* mdcModule = mdcGeomPar->getModule(sectorNum,moduleNum);
  HGeomTransform& modTransLab = mdcModule->getLabTransform();

  // get layer transformation
  HGeomVolume* layer = mdcModule->getRefVolume()->getComponent(layerNum);
  HGeomTransform& layerTrans = layer->getTransform();

  HGeomVector hitLayer, hitModule, hitLab, hitSector;

  // set x, y from Geant hit, z at entrance into volume (points 0-3)
  hitLayer.setX(x);
```

```
  hitLayer.setY(y);
  hitLayer.setZ(layer->getPoint(0)->getZ());
  hitLayer.print();

  // transform from layer to module coordinate system
  hitModule = layerTrans.transFrom(hitLayer);
  hitModule.print();

  // transform from module to LAB system
  hitLab = modTransLab.transFrom(hitModule);
  hitLab.print();

  // transform from LAB system to sector coordinate system
  hitSector = sectorTrans.transTo(hitLab);
  hitSector.print();

  // or do it all in one line
  hitSector = sectorTrans.transTo(modTransLab.transFrom(layerTrans.transFrom(hitLayer)));

  delete gHades;
}
```

# Chapter 4

# The HADES Oracle database

## 4.1 Overview

The HADES database is the central place, where information and parameters from the various systems flow together: the file catalog is filled by the DAQ and the EPICS slow control data are stored online.

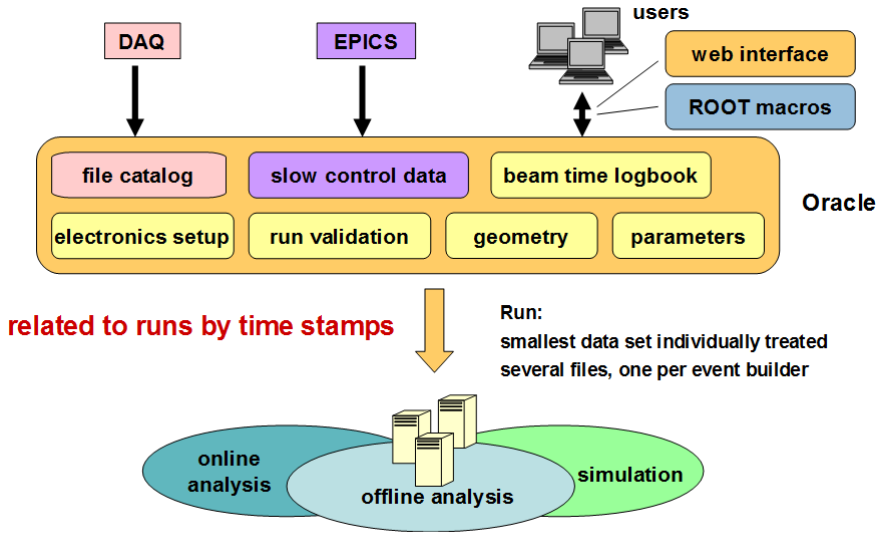



Figure 4.1: Overview

Many users store and retrieve data either by programs or web interfaces: the beam time logbook, information about the detector hardware and electronics setup, the run quality, the geometry and alignment, analysis parameters and conditions.
All these data are related by timestamps.

## 4.2 The version management of parameter containers

Most parameters used by the analysis change over time and need a version management in Oracle. Some data can have only one valid value at a certain date as for example the position of a certain cable, other ones may even change over time for a certain DAQ or simulation run depending on the status of the analysis (different generations of DST productions).

The major requirements are:

- It must be possible to get a consistent set of parameters at any date.
- To keep the history, no information, even if it is wrong, should to be overwritten without trace, to get an answer on questions like this: *"Which parameters have I used when I did the analysis of these runs six months ago?"*

- Good performance

All tree- and condition-style parameter containers have a **three dimensional version management** (fig: 4.2):

1. time axis of runs
2. time axis of history
3. parameter context

A version is valid for a certain time range on the runs axis and on the history axis (defines a 2-dim plane) and for a specific parameter context (different 2-dim planes with individual time ranges).

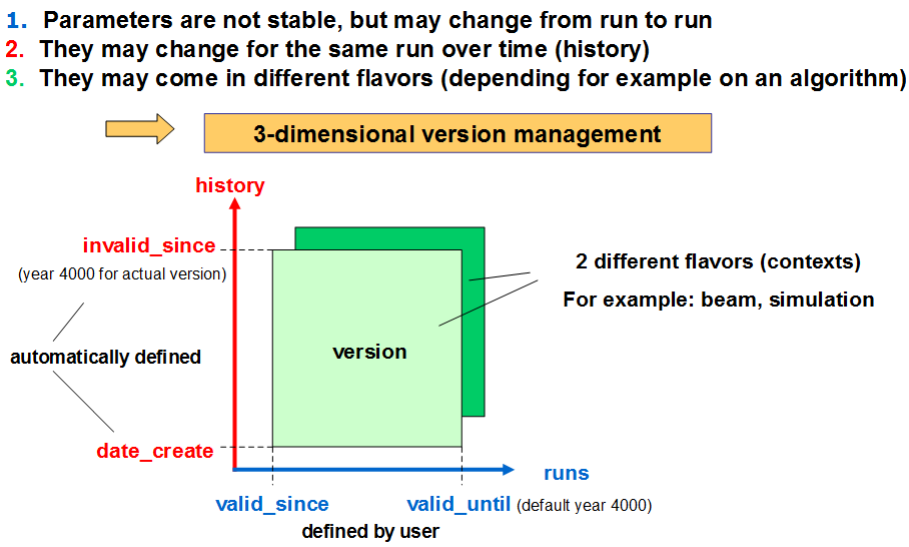



Figure 4.2: Parameters for the version management

A parameter or parameter set has 5 variables in Oracle for the version management:

| *Variable* | *Description* |
|---|---|
| valid_since | First date when the entry is valid. |
| valid_until | Last date when the entry is still valid.<br>(High date “01-JAN-4000 00:00:00“ for last version actually still valid) |
| date_create | Date when a version is inserted |
| invalid_since | Date when the entry is replaced by a new or better version and therefore gets invalid.<br>(High date “01-JAN-4000 00:00:00“ for valid version, historic dates for invalid ones) |
| context_id | Identifier of the parameter context |

Valid_since and valid_until define a time range on the runs axis. They are defined by the user in the WebDB validation form.
Date_create and invalid_since define a time range on the history axis and are set automatically by the validation interface.

Additionally the author and a description is stored for each version.

**Example:**
Figure 4.3: A user validates a version 1 of for example calibration parameters at the beginning of a beam time starting at valid_since_1. He does not specify an end of the time range, but leaves it open. The program sets the end to the January 1st in year 4000.
The analysis of every new run will now get the version 1.
After some time, after for example new pedestals have been taken, he validates a new version 2, starting at valid_since_2, and again open-end.
The web interface first changes the value of the invalid_since column in the version 1 record from year 4000 to the actual date minus one second. Then it inserts the version 1 again with the time range ending at valid_until_1. This is one second less than the new valid_since of version 2. Then it inserts the new version 2. This leads to three

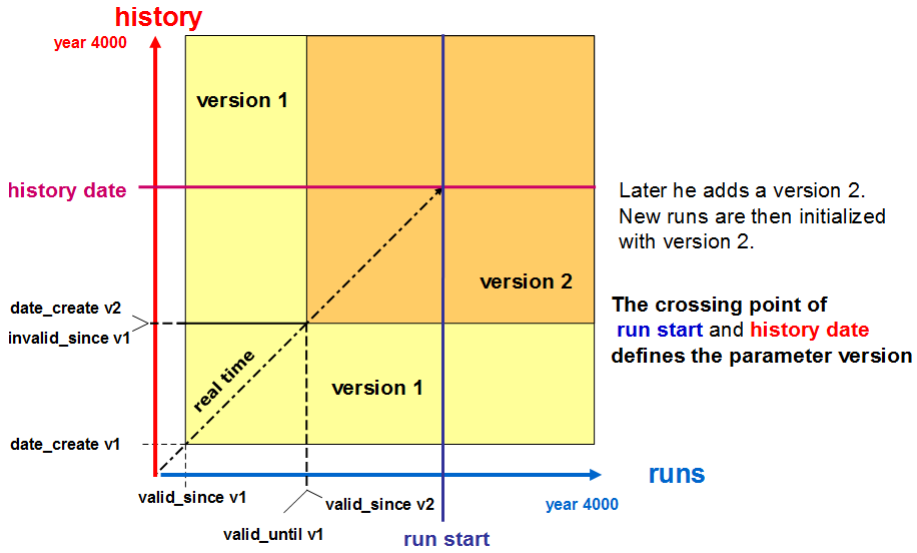



Figure 4.3: Finding the version (1)

rectangles which do not overlap.
Now every new run is initialized with version 2.
Figure 4.4: After a re-calibration maybe after the beam time, the user comes up with new version 3 and 4, which replace the old ones.

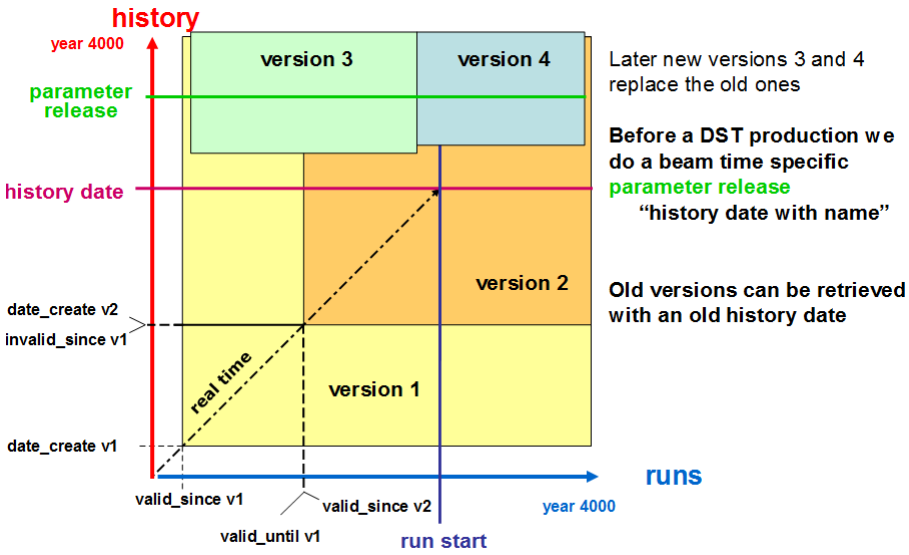



Figure 4.4: Finding the version(2)

Not all parameter containers are independent. Therefore we create before a DST production a beam time specific parameter release, which is a history date with a name stored in the data base. By default the last parameter release is used for initialization. But the user can overrule this in the macro and set any history date.

## 4.3 The WebDB Site

All web interfaces are accessible from the WebDB Site after login.

**Login:**
Direct link:
`https://hades-db.gsi.de/pls/hades_webdb/hxsite.main`
Click on Login button to get the login form.

From the HADES web page:
⇒ internal ⇒ WebDB
The login form pops up.

Login as user (not case sensitive)

**hades** This user has read-only access to most of the applications and can insert entries into the beam time log book and the shift plan of a beam time.

**xxx_oper** Each detector and the DAQ have specific access accounts to insert and change data via forms, mdc_oper for example can validate MDC parameters.

Fig. 4.5 shows the page after login.
All applications are stored in a tree-like folder structure. The navigation on the left side shows the links to the folders accessible by the user, the right side their content (here the content of the ROOT folder).

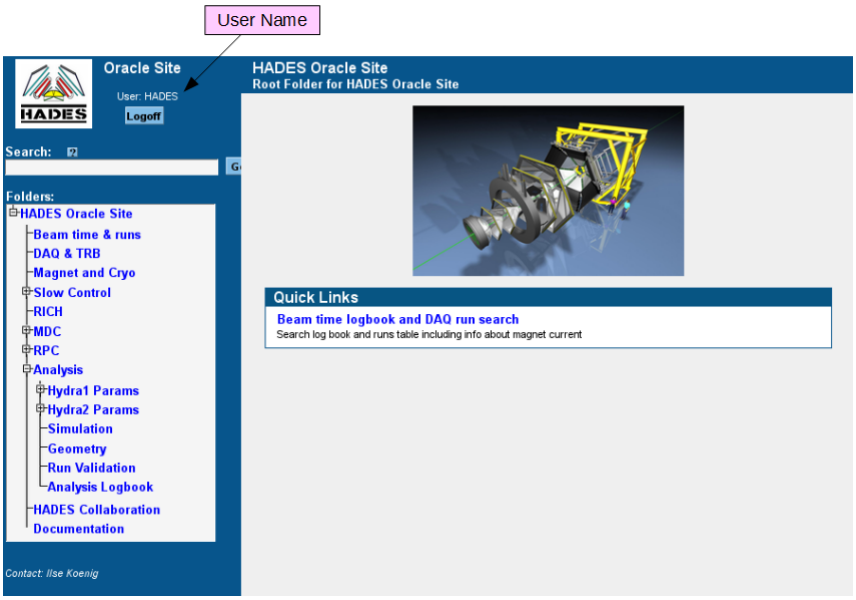



Figure 4.5: The WebDB Site

**Tip:** In the upper left part you can see the user name. The web browser typically keeps the http authentification after logout. When you switch from user hades to xxx_oper you might still be logged in as user hades. You will not see some detector specific folders or applications or get error messages ("Insufficient privileges").
Either clear the private cache or kill all browser windows and start again from a new one.
The next pages show some WebDB applications useful for analysis users (**Status at 20-JUL-2015**).

## 4.4 Experiments

The most central table in Oracle is the one for experiments. It defines a time range for DAQ runs and simulation reference runs, beam time logbook entries, shift plans and many more.

The application "**Experiment info**" in the folder "**Beamtime & runs**" allows to search for experiments. Fig. 4.6

shows the form[1] and fig. 4.7 the corresponding result. The yellow lines shows the beam times and detector tests in the HADES cave, the blue ones the simulation projects (location VIRTUAL) and the red line a beam time outside GSI[2].
The time range begin - end and the location is used to get the lists of runs in the analysis interface.

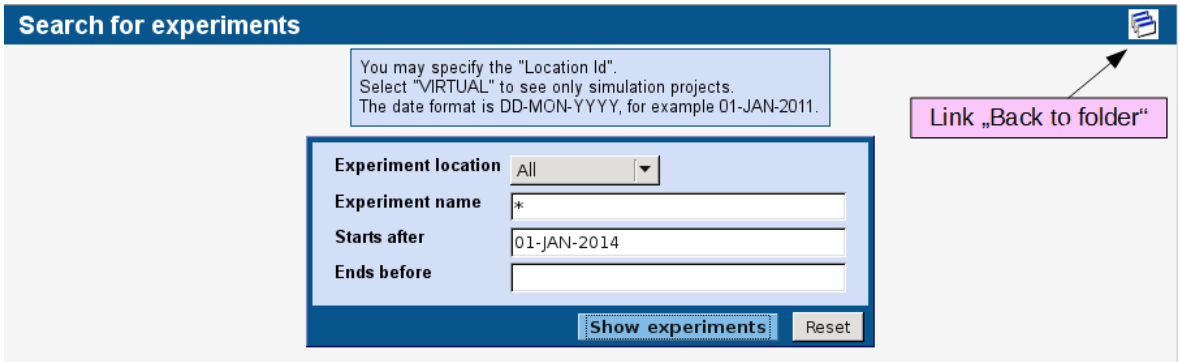
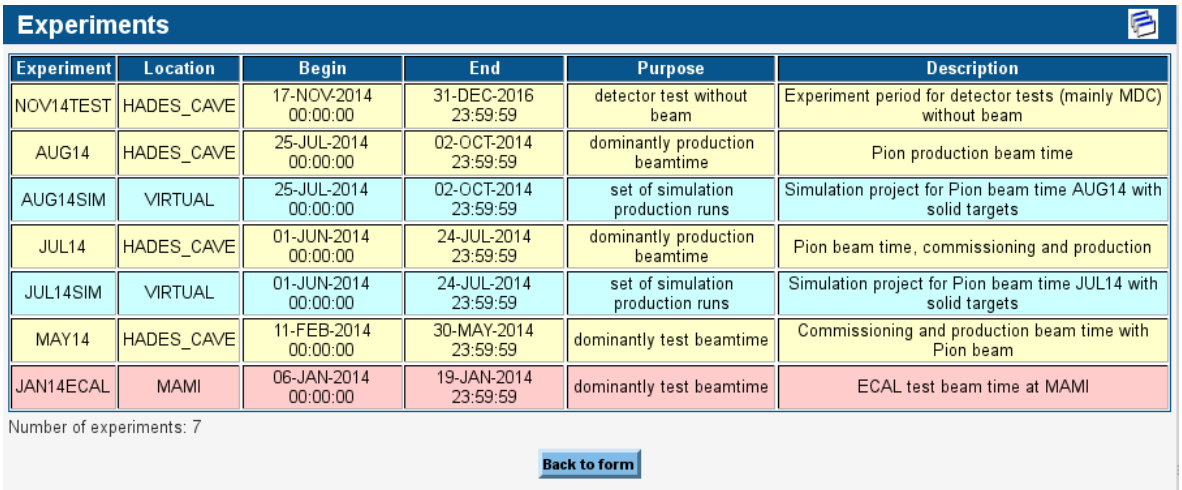



Figure 4.6: Form to search for experiments



Experiments

| Experiment | Location | Begin | End | Purpose | Description |
|---|---|---|---|---|---|
| NOV14TEST | HADES_CAVE | 17-NOV-2014 00:00:00 | 31-DEC-2016 23:59:59 | detector test without beam | Experiment period for detector tests (mainly MDC) without beam |
| AUG14 | HADES_CAVE | 25-JUL-2014 00:00:00 | 02-OCT-2014 23:59:59 | dominantly production beamtime | Pion production beam time |
| AUG14SIM | VIRTUAL | 25-JUL-2014 00:00:00 | 02-OCT-2014 23:59:59 | set of simulation production runs | Simulation project for Pion beam time AUG14 with solid targets |
| JUL14 | HADES_CAVE | 01-JUN-2014 00:00:00 | 24-JUL-2014 23:59:59 | dominantly production beamtime | Pion beam time, commissioning and production |
| JUL14SIM | VIRTUAL | 01-JUN-2014 00:00:00 | 24-JUL-2014 23:59:59 | set of simulation production runs | Simulation project for Pion beam time JUL14 with solid targets |
| MAY14 | HADES_CAVE | 11-FEB-2014 00:00:00 | 30-MAY-2014 23:59:59 | dominantly test beamtime | Commissioning and production beam time with Pion beam |
| JAN14ECAL | MAMI | 06-JAN-2014 00:00:00 | 19-JAN-2014 23:59:59 | dominantly test beamtime | ECAL test beam time at MAMI |

Number of experiments: 7

Back to form

Figure 4.7: List of experiments in specified time range

### 4.4.1 DAQ runs

DAQ runs are stored in Oracle online by Perl scripts and can be shown in the beam time logbook. Not shown there are the list of files (one for each event builder) belonging to the run.
The master event builder defines the run id (time of the first event in seconds since 01-JAN-1970 minus $1.2 \times 10^9$) common for all event builder files. The smallest start time of all files with non-zero events is stored as start time of this run. All files with the same run id use this start time for the initialization of the analysis parameters.
The common run type is defined by the file prefix, for example run type BEAM by prefix `be`, COSMIC by `co`.

The application "**Charts of beam time and run statistics**" in folder "**Beamtime &runs**" (topic Monitoring) allows to show the statistic for beam time runs and the corresponding event builder files.
After selecting a beam time one gets the form shown in the upper part of fig. 4.8. See the HELP for instructions to filter the data and to produce different bar charts (for example number of runs, number of events, ...).
The lower part in the figure shows the result of the selection grouped by day.
The color indicates the mean magnet current (always averaged over the time range of one bar).
If one clicks on the day link one would get the results by hour and if one clicks there on an hour link one gets the list of files with non-zero events. Fig. 4.9 shows an example.

[1] Some applications are shown in the full window. The icon link "Back to folder" leads back the the folder page with the navigation bar on the left side.
[2] To use the beam time logbook a beam time must be defined.

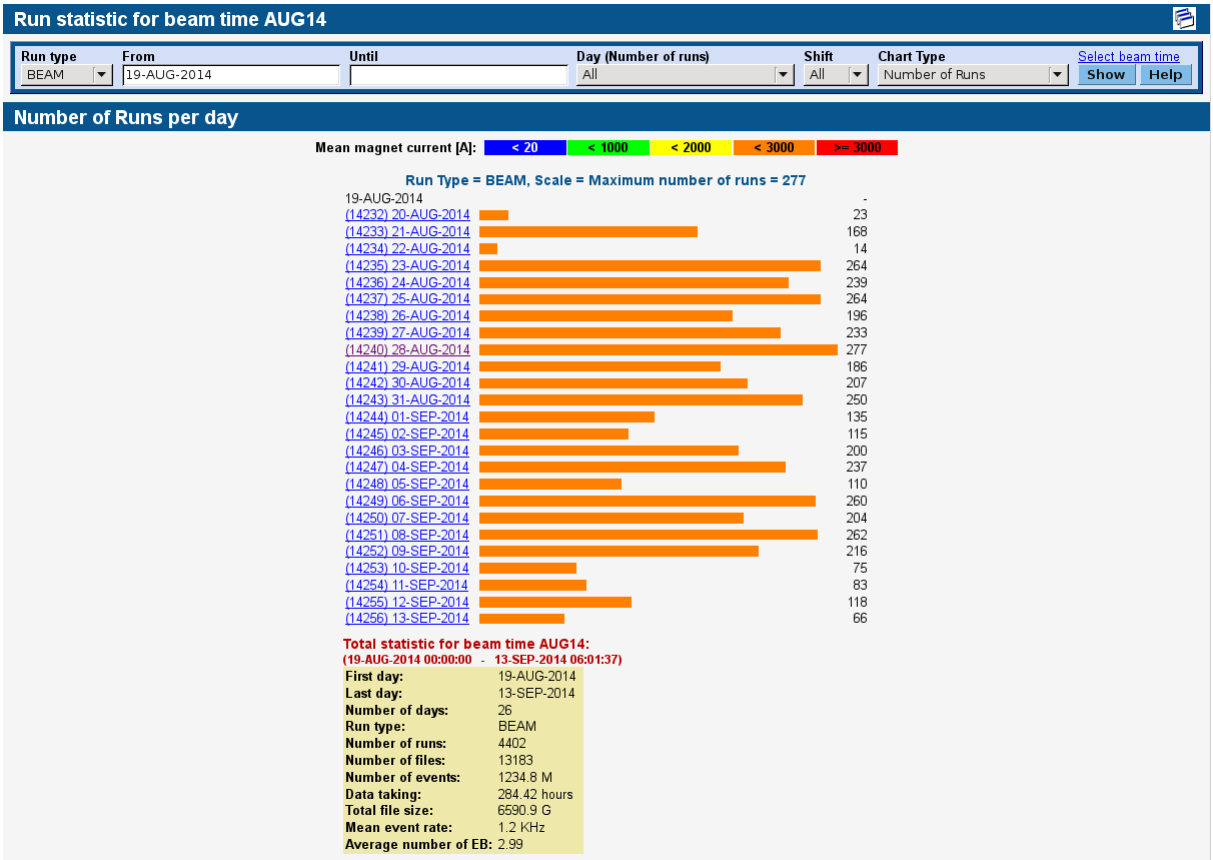
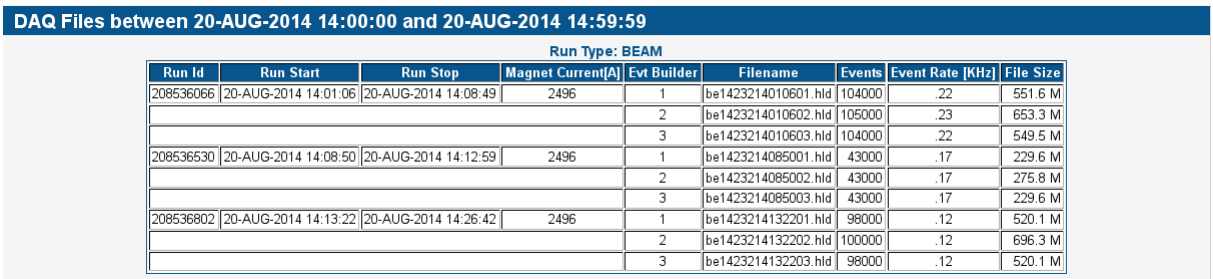



Figure 4.8: Beam time and run statistic



DAQ Files between 20-AUG-2014 14:00:00 and 20-AUG-2014 14:59:59

Run Type: BEAM

| Run Id | Run Start | Run Stop | Magnet Current[A] | Evt Builder | Filename | Events | Event Rate [KHz] | File Size |
|---|---|---|---|---|---|---|---|---|
| 208536066 | 20-AUG-2014 14:01:06 | 20-AUG-2014 14:08:49 | 2496 | 1 | be1423214010601.hld | 104000 | .22 | 551.6 M |
| | | | | 2 | be1423214010602.hld | 105000 | .23 | 653.3 M |
| | | | | 3 | be1423214010603.hld | 104000 | .22 | 549.5 M |
| 208536530 | 20-AUG-2014 14:08:50 | 20-AUG-2014 14:12:59 | 2496 | 1 | be1423214085001.hld | 43000 | .17 | 229.6 M |
| | | | | 2 | be1423214085002.hld | 43000 | .17 | 275.8 M |
| | | | | 3 | be1423214085003.hld | 43000 | .17 | 229.6 M |
| 208536802 | 20-AUG-2014 14:13:22 | 20-AUG-2014 14:26:42 | 2496 | 1 | be1423214132201.hld | 98000 | .12 | 520.1 M |
| | | | | 2 | be1423214132202.hld | 100000 | .12 | 696.3 M |
| | | | | 3 | be1423214132203.hld | 98000 | .12 | 520.1 M |

Figure 4.9: DAQ runs and files

### 4.4.2 Simulation projects and reference runs

The version management in Oracle is based on time ranges. Each DAQ run has a start time and this time stamp (together with the history date and the context) defines the version used for the initialization of this run. Runs are taken in consecutive order and have no time overlap.
On the other hand simulations are done in parallel for different beam times or proposals for future experiments and the real start time is not a good ordering parameter.
To use the same tables/views and interfaces in Oracle it was necessary to give simulation runs artificial timestamps and to group them in projects and sub-projects.[3]

A **simulation project** is typically a simulation for a beam time (example APR12SIM) and has also the same time range. But to do simulations for future experiments, for example to simulate the replacement of the Shower detector by the ECAL, and to store dedicated geometry setups for it, some projects have been added with intermediate time ranges (example ECAL13SIM).

[3]The additional grouping in generations was discarded for simulation projects since 2010.

Each project contains **sub-projects** with fixed

- projectile + target system
- projectile energy
- field setting
- target

Each sub-project has an artificial time range of one day. It has at least one **reference run** with a descriptive name and a run id easy to remember. The first reference run typically starts at midnight and has a duration of only one second.
The run id is used in the analysis as reference run for the initialization of parameter containers, while the name is used in the HGeant2 configuration file to read the geometry (see 3.5.1).

Fig. 4.10 shows the WebDB application in folder "**Analysis** ⇒ **Simulation**" which allows to get the list of reference runs for a selected simulation project, here as example for the simulations for the Pion beam time in August 2014.

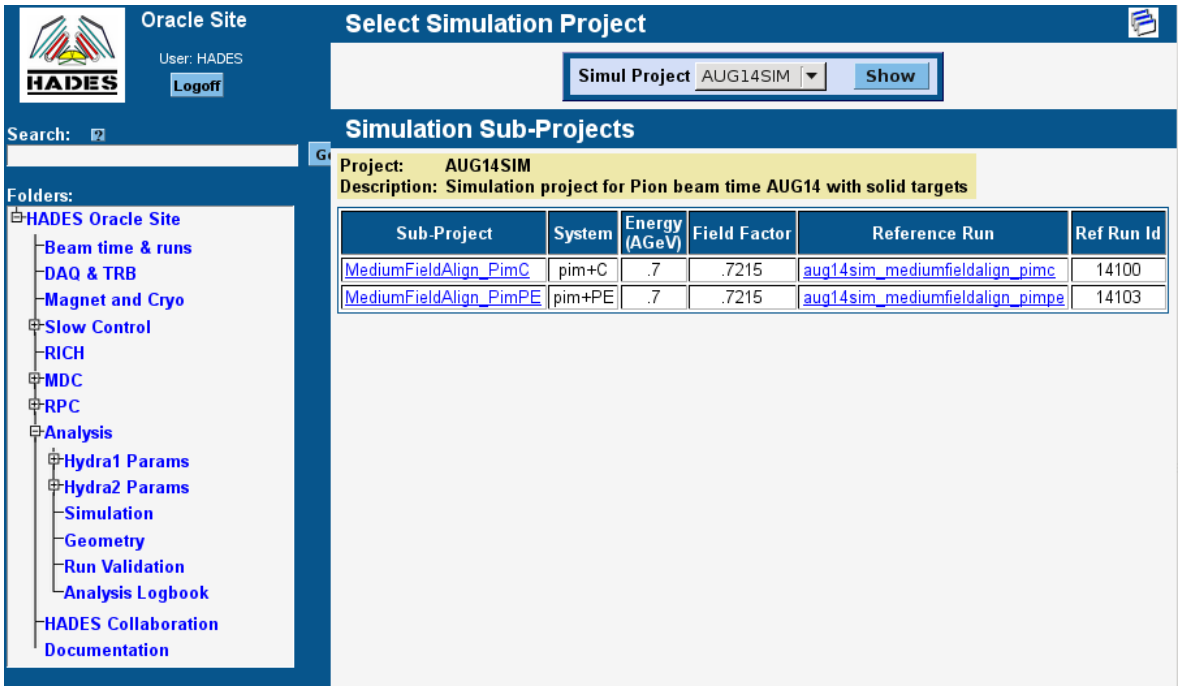



Figure 4.10: Simulation sub-projects and reference runs

If people validate parameters for simulation they use links in pop-up windows showing the projects and sub-projects names. A click on the link adds the corresponding time in the form, time begin for valid_since, time end for valid_until.
Also in the public interface typically only the names of the reference runs are visible, not the time stamps.

Because the time stamps are so artificial and the rules hard to remember, a WebDB interface was developed to make it more comfortable to add projects and sub-projects and to avoid mistakes. The link is in the same folder but one needs to login as a special user to execute it.

## 4.5 The WebDB folder Hydra2 Params

For the change from Hydra1 to Hydra2 a major revision of the Oracle production accounts was done in 2009. Backward compatibility was not required. Many new accounts were created with partially different table design. Only the last versions of the parameters were copied into these tables and validated since 01-JAN-2010. The Hydra2 analysis interface cannot read Hydra1 parameter containers and vice versa.
Many web interfaces were cleaned and modernized (new framework with CSS and Javascript). The old Hydra1 web interfaces were not touched and still use the old layout.

The folder "**Analysis** ⇒ **Hydra2 Params**" (Fig. 4.11) contains the applications related to parameter containers in Hydra2. It contains also sub-folders for tree- and condition-style parameter containers.

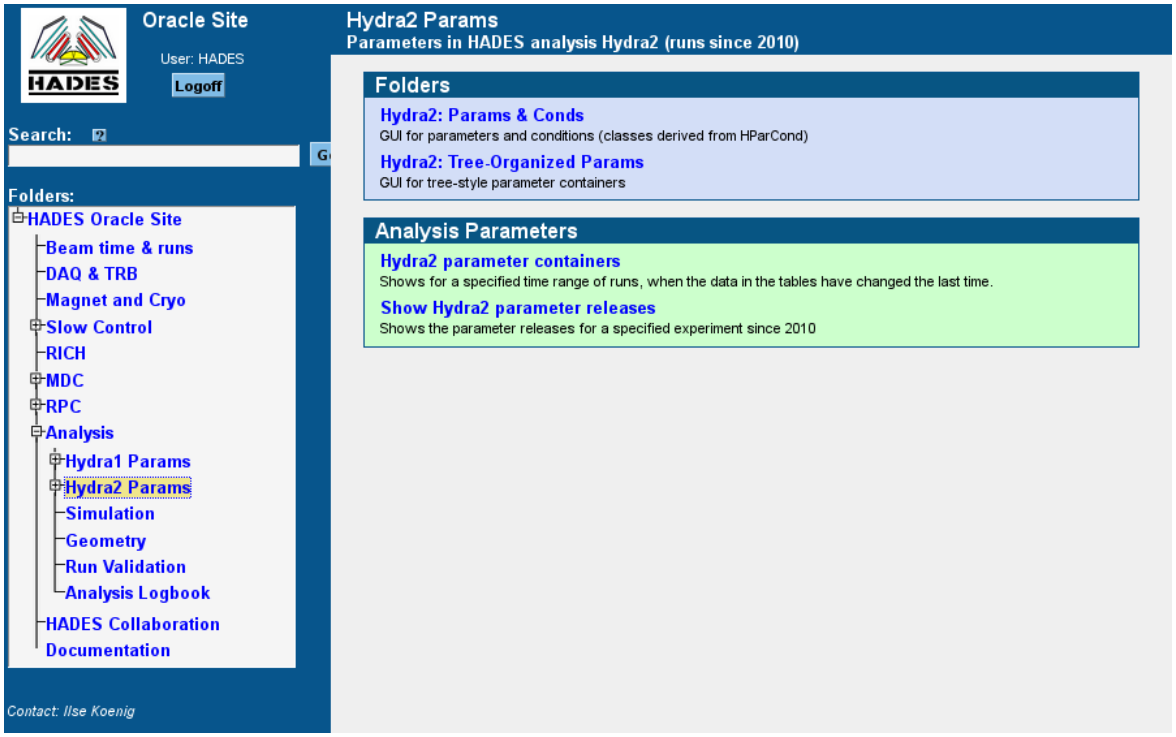



Figure 4.11: The WebDB folder Analysis ⇒ Hydra2 Params

### 4.5.1 Parameter releases

Fig. 4.12 shows an example of the application "**Show Hydra2 parameter releases**".
As first step one fills the form in the upper part and selects one or more beam times and/or simulation projects. If one then clicks on `Show` one gets the list of all parameter releases. The green line shows the current one, which would be used for initialization if no history date it set explicitly.

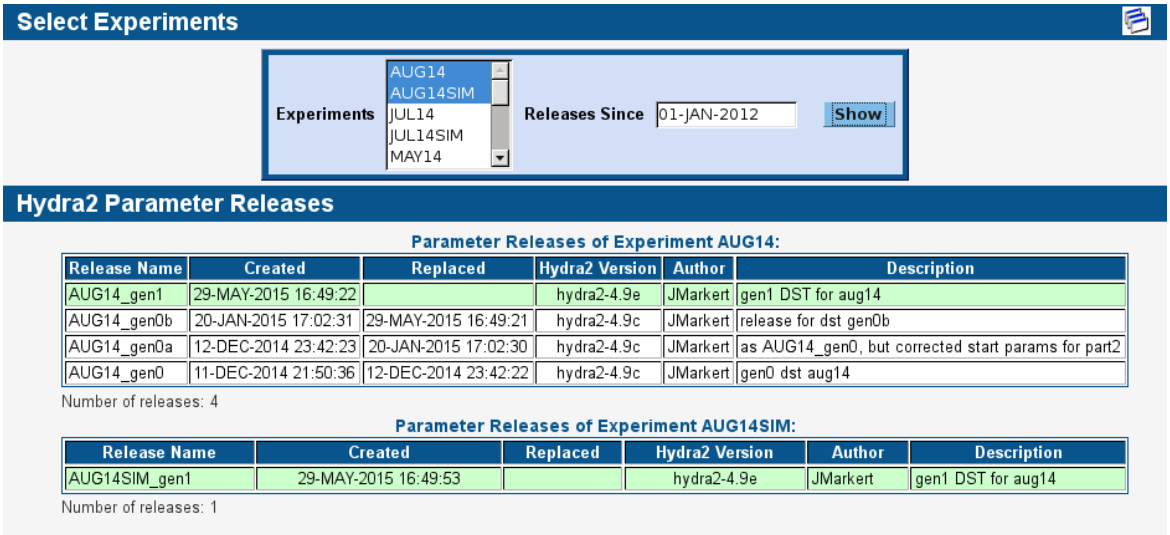



Figure 4.12: Hydra2 parameter releases

### 4.5.2 Overview of parameter containers

Fig. 4.13 shows an example of the application "**Hydra2 parameter containers**".
For a selected beam time it lists all parameter containers (still missing: geometry parameter containers) with the

time stamp of the last change, in red the ones which changed since the last parameter release[4].

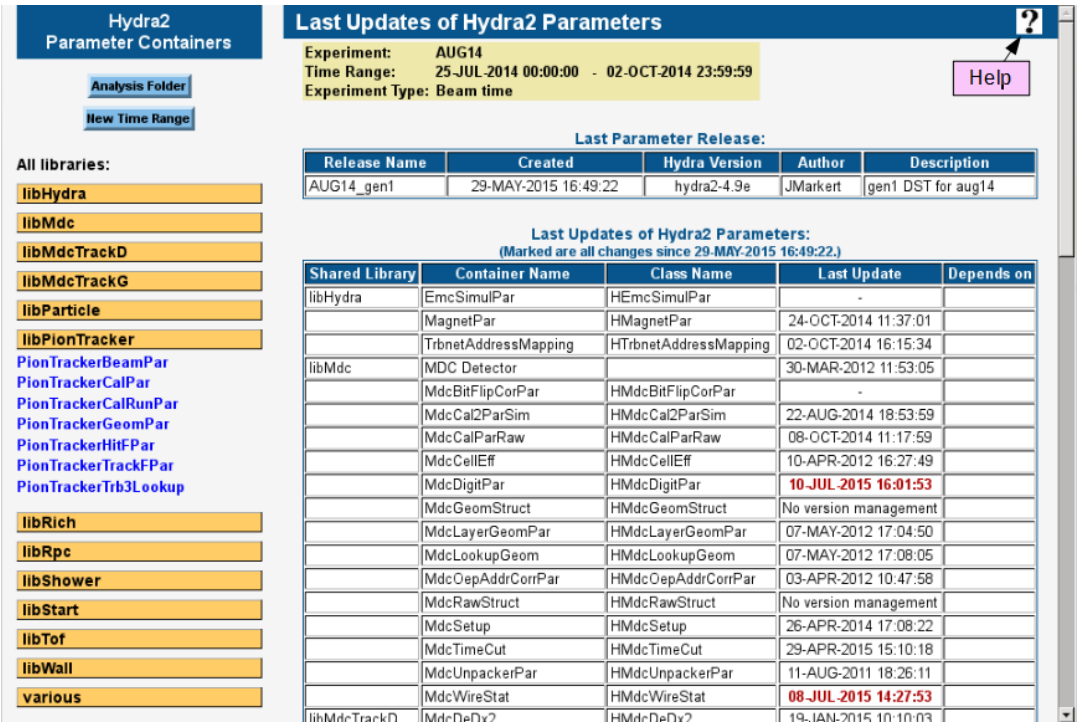



Figure 4.13: Hydra2 parameter containers overview

The navigation bar on the left side provides access to the parameter containers grouped by the shared libraries. Click on a shared library to see the links to the data pages for the parameter containers in this library.

## 4.6 Condition-style parameter containers

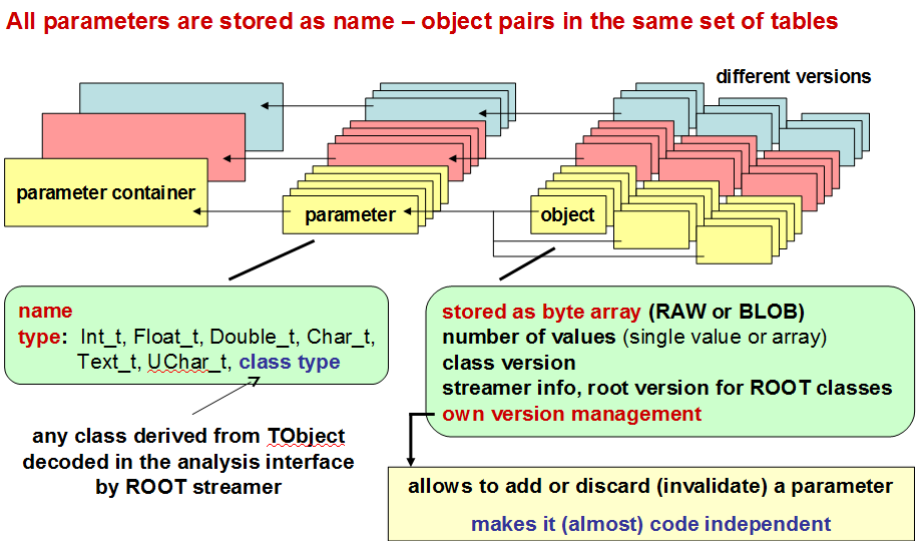



Figure 4.14: Storage of condition-style parameter containers

Each parameter belongs to a parameter container, has a name and a type. Possible types are the basic ROOT c-types, but also any class derived from TObject.
Different data objects (= different versions) exist for each parameter. The data are stored as byte array either in

[4]In the entrance form one may set a date to highlight all later changes. This might be usesful if one created a parameter ROOT file once and wants to see which parameters changed meanwhile.

a RAW column for single number values and small arrays or in a binary large object (BLOB). Also stored is the number of values and for classes the class version, for ROOT classes additionally the streamer info and the ROOT version for backward compatibility. Basic types can be decoded in the web interface, classes only in the c++ interface with the ROOT streamer.

Each data object has its own version management, which allows to add a new parameter or to discard an old parameter no longer needed without any change of the other parameters. The analysis interface reads all parameters valid for the specified run and history date and stores them in a list. The parameter container takes from the list only the parameters (= data members) in the current class version. This makes the interface code independent.

### 4.6.1 Search for parameter sets

Fig. 4.15 shows the applications in the folder "**Hydra2 Params ⇒ Params & Conds**" for condition-style parameter containers.

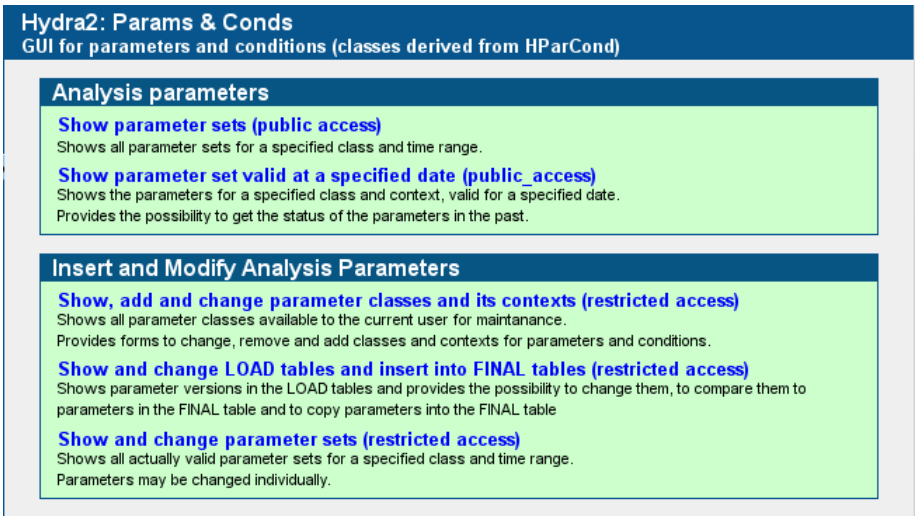



Figure 4.15: Folder Hydra2 Params ⇒ Params & Conds

The application "**Show parameter sets**" allows to search for the parameter sets of a selected parameter container and experiment. Fig. 4.16 shows the result for the parameter container `MdcDigiPar` and beam time AUG14.[5] The program searches for all valid parameters in the beam time range. If one or more parameters change in this time range one will see more than one set. The time range of one set is the maximum of the valid_since of the parameters and the minimum of all valid_until.

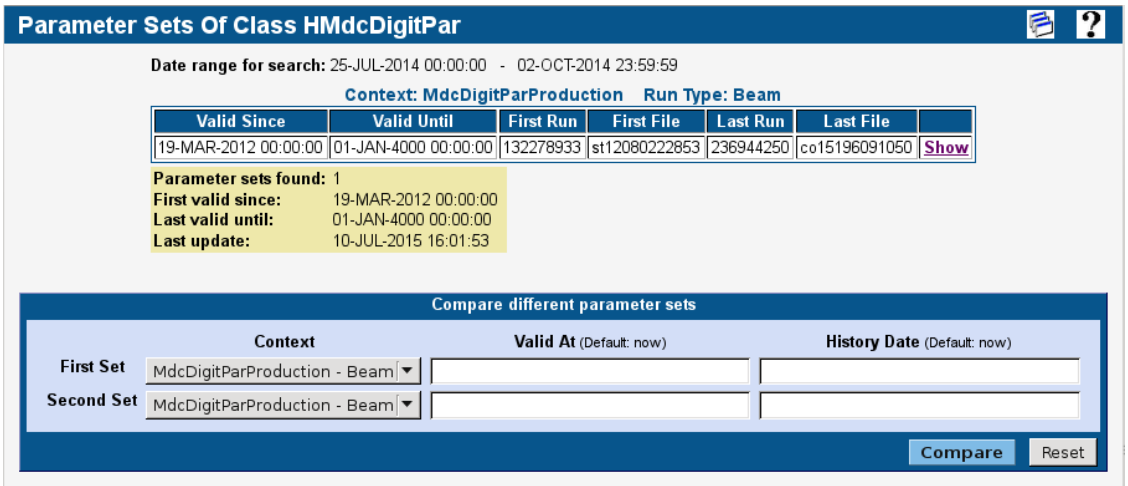



Figure 4.16: Parameters sets of class HMdcDigiPar

[5] For event embedding simulation parameters must be also validated for beam runs.

The link `Show` displays the valid data, the time stamps of the version management and a link to display the comment (fig. 4.17). Some columns may be sorted, here for `Date Create`.

**Valid Parameter Values**
**Class: HMdcDigitPar Context: MdcDigitParProduction Run Type: Beam**

All columns with up/down icons may be sorted.

| Param Id | Param Name | Type | Value | Num Values | Streamer Info | Valid Since | Valid Until | Date Create | Comment Id |
|---|---|---|---|---|---|---|---|---|---|
| 5160 | layEff | Float_t | 9.5300E-01 9.6000E-01 9.7500E-01 9.7600E-01 9.6300E-01<br>9.5300E-01 9.7100E-01 9.8100E-01 9.8700E-01 9.8500E-01<br>9.8400E-01 9.7800E-01 9.8300E-01 9.8900E-01 9.8700E-01<br>9.8500E-01 9.8800E-01 9.8100E-01 9.8900E-01 9.8600E-01<br>9.8400E-01 9.8700E-01 9.8400E-01 9.9000E-01 9.5100E-01<br>... Show all | 144 | | 19-MAR-2012 00:00:00 | 01-JAN-4000 00:00:00 | 10-JUL-2015 16:01:53 | 6523 |
| 6160 | layEffScale | Float_t | 9.6800E-01 9.9100E-01 9.9000E-01 9.8900E-01 9.9300E-01<br>9.7300E-01 9.2900E-01 9.5800E-01 9.5200E-01 9.6900E-01<br>9.4300E-01 9.0300E-01 9.9700E-01 9.8700E-01 1.0000E+00<br>1.0000E+00 9.9700E-01 9.7100E-01 9.5400E-01 9.7000E-01<br>9.8500E-01 9.8500E-01 9.8100E-01 9.6700E-01 9.7700E-01<br>... Show all | 144 | | 19-MAR-2012 00:00:00 | 01-JAN-4000 00:00:00 | 10-JUL-2015 16:01:53 | 6523 |
| 6156 | scaleTime1ErrMIPS | Float_t | 1.0000E+00 1.0000E+00 1.0000E+00 1.0000E+00 | 4 | | 19-MAR-2012 00:00:00 | 01-JAN-4000 00:00:00 | 15-OCT-2014 10:07:27 | 6462 |
| 6158 | scaleTime1Err | Float_t | 5.0000E-01 5.0000E-01 5.7000E-01 5.7000E-01 | 4 | | 19-MAR-2012 00:00:00 | 01-JAN-4000 00:00:00 | 15-OCT-2014 10:07:27 | 6462 |
| 6154 | cellEffScale | Float_t | 7.0000E-01 | 1 | | 19-MAR-2012 00:00:00 | 01-JAN-4000 00:00:00 | 18-DEC-2013 11:43:32 | 6152 |
| 5162 | signalspeed | Float_t | 4.0000E-03 | 1 | | 01-JAN-2010 00:00:00 | 01-JAN-4000 00:00:00 | 15-JUL-2010 17:50:44 | 5159 |

**Number of parameters:** 6
**Valid for date range:** 19-MAR-2012 00:00:00 - 01-JAN-4000 00:00:00
**Status at:** 17-JUL-2015 12:38:10

**Compare different parameter sets**

| | Context | Valid At (Default: now) | History Date (Default: now) |
|---|---|---|---|
| First Set | MdcDigitParProduction - Beam | 19-MAR-2012 00:00:00 | 10-JUL-2015 |
| Second Set | MdcDigitParProduction - Beam | | |

Compare Reset

Figure 4.17: Parameters of class HMdcDigiPar

The form in fig. 4.17 allows to compare parameter sets. In this example one will see the differences of the data compared to the historic data valid before the last validation at `10-JUL-2015 16:01:53` (fig. 4.18).

**Compare Parameter Sets for Class HMdcDigitPar**

**Sets**

| Set | Context | Valid At | Status At |
|---|---|---|---|
| 1 | MdcDigitParProduction - Beam | 19-MAR-2012 00:00:00 | 10-JUL-2015 00:00:00 |
| 2 | MdcDigitParProduction - Beam | 17-JUL-2015 12:39:10 | 17-JUL-2015 12:39:10 |

**Parameters in both sets:**

| Param Id | Param Name | Type | Value | Num Values | Streamer Info | Valid Since | Valid Until | Date Create | Invalid Since | Comment Id |
|---|---|---|---|---|---|---|---|---|---|---|
| 5160 | layEff | Float_t | 9.5300E-01 9.6100E-01 9.7500E-01 9.7700E-01 9.6400E-01<br>9.4500E-01 9.6800E-01 9.7900E-01 9.8400E-01 9.8000E-01<br>9.8500E-01 9.7600E-01 9.8300E-01 9.8900E-01 9.8500E-01<br>9.8400E-01 9.8400E-01 9.8000E-01 9.8400E-01 9.8400E-01<br>9.8100E-01 9.8300E-01 9.8700E-01 9.9600E-01 9.5300E-01<br>... Show all | 144 | | 19-MAR-2012 00:00:00 | 01-JAN-4000 00:00:00 | 15-OCT-2014 10:07:27 | 10-JUL-2015 16:01:52 | 6462 |
| | | Float_t | 9.5300E-01 9.6000E-01 9.7500E-01 9.7600E-01 9.6300E-01<br>9.5300E-01 9.7100E-01 9.8100E-01 9.8700E-01 9.8500E-01<br>9.8400E-01 9.7800E-01 9.8300E-01 9.8900E-01 9.8700E-01<br>9.8500E-01 9.8800E-01 9.8100E-01 9.8900E-01 9.8600E-01<br>9.8400E-01 9.8700E-01 9.8400E-01 9.9000E-01 9.5100E-01<br>... Show all | 144 | | 19-MAR-2012 00:00:00 | 01-JAN-4000 00:00:00 | 10-JUL-2015 16:01:53 | 01-JAN-4000 00:00:00 | 6523 |
| 5162 | signalspeed | Float_t | 4.0000E-03 | 1 | | 01-JAN-2010 00:00:00 | 01-JAN-4000 00:00:00 | 15-JUL-2010 17:50:44 | 01-JAN-4000 00:00:00 | 5159 |
| | | Same in both sets | | | | | | | | |
| 6154 | cellEffScale | Float_t | 7.0000E-01 | 1 | | 19-MAR-2012 00:00:00 | 01-JAN-4000 00:00:00 | 18-DEC-2013 11:43:32 | 01-JAN-4000 00:00:00 | 6152 |
| | | Same in both sets | | | | | | | | |
| 6156 | scaleTime1ErrMIPS | Float_t | 1.0000E+00 1.0000E+00 1.0000E+00 1.0000E+00 | 4 | | 19-MAR-2012 00:00:00 | 01-JAN-4000 00:00:00 | 15-OCT-2014 10:07:27 | 01-JAN-4000 00:00:00 | 6462 |
| | | Same in both sets | | | | | | | | |
| 6158 | scaleTime1Err | Float_t | 5.0000E-01 5.0000E-01 5.7000E-01 5.7000E-01 | 4 | | 19-MAR-2012 00:00:00 | 01-JAN-4000 00:00:00 | 15-OCT-2014 10:07:27 | 01-JAN-4000 00:00:00 | 6462 |
| | | Same in both sets | | | | | | | | |
| 6160 | layEffScale | Float_t | 9.8000E-01 9.9500E-01 9.9400E-01 9.9100E-01 1.0000E+00<br>9.7600E-01 9.3300E-01 9.5800E-01 9.5600E-01 9.7400E-01<br>9.4700E-01 9.0600E-01 9.9900E-01 9.8800E-01 1.0000E+00<br>1.0000E+00 9.9300E-01 9.7500E-01 9.5700E-01 9.7100E-01<br>9.8600E-01 9.8900E-01 9.8300E-01 9.8000E-01 9.8300E-01<br>... Show all | 144 | | 19-MAR-2012 00:00:00 | 01-JAN-4000 00:00:00 | 15-OCT-2014 10:07:27 | 10-JUL-2015 16:01:52 | 6462 |
| | | Float_t | 9.6800E-01 9.9100E-01 9.9000E-01 9.8900E-01 9.9300E-01<br>9.7300E-01 9.2900E-01 9.5800E-01 9.5200E-01 9.6900E-01<br>9.4300E-01 9.0300E-01 9.9700E-01 9.8700E-01 1.0000E+00<br>1.0000E+00 9.9700E-01 9.7100E-01 9.5400E-01 9.7000E-01<br>9.8500E-01 9.8500E-01 9.8100E-01 9.6700E-01 9.7700E-01<br>... Show all | 144 | | 19-MAR-2012 00:00:00 | 01-JAN-4000 00:00:00 | 10-JUL-2015 16:01:53 | 01-JAN-4000 00:00:00 | 6523 |

**Number of parameters:** 6
**Number of different values:** 2

Figure 4.18: Result of comparison

In this example the first set is the historic one, the second the actual valid set. As `Valid At` for the first set, one chooses the last `Valid Since` of all parameters. As `History Date` one puts a time stamp at least one second earlier as the last `Date Create` (if omitted the time part is set to 00:00:00).
Fig. 4.18 shows the result. The differences are shown in red. As expected two parameters were changed.

For event embedding in the analysis it is essential that the parameters needed by the digitizer are the same for beam and simulation runs. To check this one can compare the parameters by choosing different contexts in the form, but the same `Valid At` and `History Date`. The parameter values should be the same, only the `Date Create` is typically different, because two validations, one for beam and one for simulation runs were done.

### 4.6.2 Adding a new class

The folder "**Hydra2 Params ⇒ Params & Conds**" contains also the non-public applications to store, validate and change condition-style parameter containers.
The privilege to manage the data is granted by library. For example the account TOF_OPER can manage parameter containers in libTof (and also libStart). The drop-down menus only show classes or libraries the user logged in can manage.
If one creates a new condition-style parameter container in Hydra2 one must add this class and its context in Oracle before one can write the parameters into the database. The application "**Show, add and change parameter classes and its contexts (restricted access)**" lists all parameter containers the user can manage and below a button `Add New Class`. It leads to a form where one selects the library, specifies the class name, the name of the parameter container, the standard context name and the run type(s) for the context (beam runs, simulations runs or both).
In the list of parameter containers it shows up as a red line. This means, one can write the parameters to Oracle but not validate them (no link). First Jochen Markert or Ilse Koenig must **confirm** the new class. As long as not validated the class and the data can be deleted from Oracle, once validated this is not allowed anymore (and very complicated). The confirmation is a kind of security check to avoid the storage of parameter containers not accepted in Hydra2 and code not committed to SVN.

### 4.6.3 Parameter validation

If one writes a condition-style parameter container into Oracle, the data are **not** stored in the final tables but in intermediate so-called LOAD tables. Here parameters can be changed, even new parameters added. After validation the data can be removed from the LOAD tables.[6]

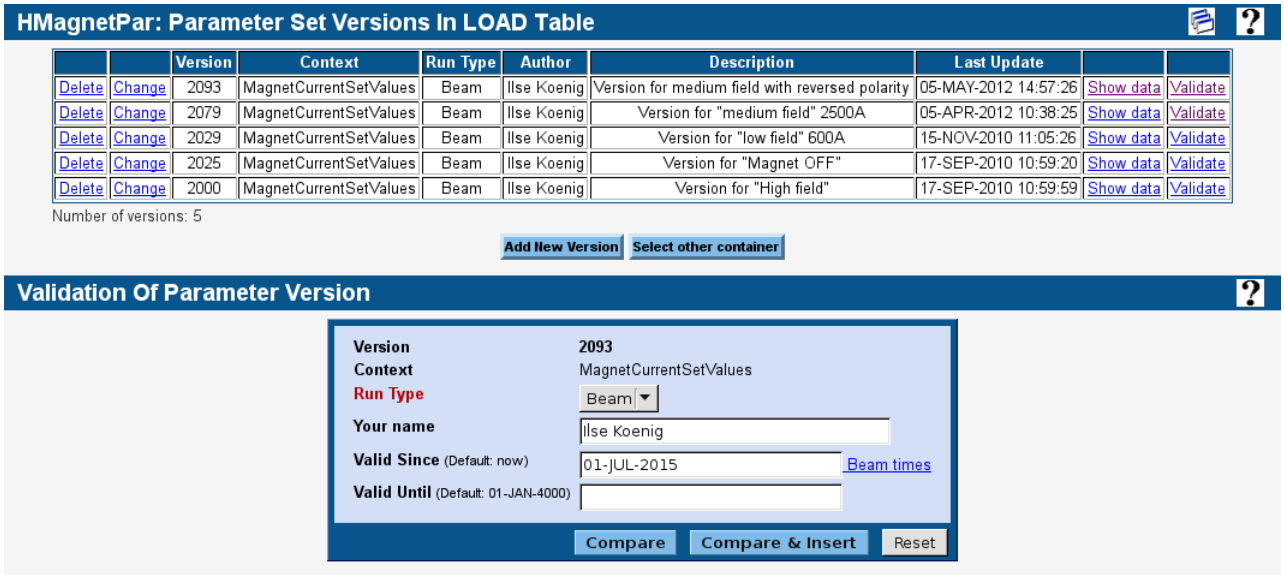



Figure 4.19: Show and validate parameters in the LOAD tables

The application "**Show and change LOAD tables and insert into FINAL tables**" shows all sets in the LOAD tables for a selected parameter container, in the upper part of figure 4.19 as example for the parameter container

[6] One should only keep standard versions in the LOAD tables, which one must validate more often for different time ranges, for example different versions of MagnetPar with the standard current settings. All others should be deleted to avoid mistakes.

MagnetPar. The link `Validate` of the uppermost version leads to the validation form below. Here one chooses the run type (Beam or Simul) and specifies the validity time range.
The link `Beam times` lists all beam times with their begin and end dates as links. If one clicks on a link the corresponding date is transferred to the form field.
For more details read the `Help`.

Figure 4.20 shows the result if one clicks on the button `Compare & Insert`. The program loops over all parameters, for each compares the new value with the value(s) valid in the specified time range and shows a preview of all changes.
Figure 4.21 on page 50 shows the corresponding `Help` page with a detailed description.

**Compare Version 2093**
**Class: HMagnetPar Context: MagnetCurrentSetValues Run Type: Beam**

**Comment for new parameters:**

| | |
|---|---|
| **Author:** | Ilse Koenig |
| **Description:** | Version for medium field with reversed polarity |

**Comparison of new and old values:**

| Status | Action | Name | Valid Since | Valid Until | Invalid Since | Date Create | Comment Id | Type | Value | Num Values | Streamer Info |
|---|---|---|---|---|---|---|---|---|---|---|---|
| New | | current | 01-JUL-2015 00:00:00 | 01-JAN-4000 00:00:00 | 01-JAN-4000 00:00:00 | now | new | Int_t | -2500 | 1 | |
| Old | | current | 13-SEP-2014 14:24:35 | 01-JAN-4000 00:00:00 | 01-JAN-4000 00:00:00 | 13-SEP-2014 14:24:35 | 6438 | Int_t | 0 | 1 | |
| Old | Update | current | 13-SEP-2014 14:24:35 | 01-JAN-4000 00:00:00 | now | 13-SEP-2014 14:24:35 | 6438 | Int_t | 0 | 1 | |
| Old | Insert | current | 13-SEP-2014 14:24:35 | 30-JUN-2015 23:59:59 | 01-JAN-4000 00:00:00 | 13-SEP-2014 14:24:35 | 6438 | Int_t | 0 | 1 | |
| New | Insert | current | 01-JUL-2015 00:00:00 | 01-JAN-4000 00:00:00 | 01-JAN-4000 00:00:00 | now | new | Int_t | -2500 | 1 | |

Number of existing parameters: 1

**Number of inserts/updates: 3**

Cancel Insert

Click on **"Insert"** to store the changes permanently
or on **"Cancel"** to release the table lock without any changes.

Figure 4.20: Preview of validation for condition-style parameter containers

**How to get a valid parameter set into the LOAD tables**

The application "**Show and change parameter sets**" does not only allow to change the validity time ranges of valid parameter sets or individual parameters (see `Help` for details) but also provides a form to get a valid set into the LOAD tables without the need to run a macro in Hydra2.

Recipe:
In the form select the parameter container and the beam time or simulation project (or specify a longer time range).
Click on the button `Show Sets`.
This shows all valid parameter sets. Click on the `Show` link of the version you want to copy.
It shows the data of all parameters in this set and below a form to copy the data. Specify the author and the comment and click on `Copy Set`.

## 4.7 Tree-style parameter containers

Although the storage of tree-style parameters is completely different the WebDB interface is quite similar to condition-style parameter containers.
The main difference comes from the fact, that the analysis interface creates a version and writes the data of this version directly into the final tables.
The values are stored as rows, each consisting of the version id, an address identifier (for example a cell id) and the values of all data members in separate columns (similar to the ASCII file). All rows with the same version id build one parameter set.
The validation interface validates the version number. It simply compares the version numbers with the ones actually valid in the specified time range. The parameters themselves are **not** compared.

All interfaces are in the folder **Hydra2 Params** ⇒ **Tree-Organized Params**.

This page show the new parameters, all old parameters actually valid in the specified time range and the actions (inserts/updates), which would be made, when this version would be inserted as the new valid version in the specified time range.

The following colors are used for a quick overview:

- Parameters in the **new** set
- **Original old** parameters (actual still valid)
- **Actions in the database**.

**Variables in the tables:**

| | |
|---|---|
| **Status:** | **New** or **Old** parameter |
| **Action:** | Action in the database: Insert or Update |
| **Name:** | Name of the parameter |
| **Valid Since:** | First time, the parameter gets valid |
| **Valid Until:** | Last time, the parameter is still valid |
| **Invalid Since:** | Time, the entry gets invalid |
| **Date Create:** | Date, the entry is (will be) created |
| **Comment Id:** | Id of the parameter comment |
| **Type:** | Type of the parameter<br>(Type of the array elements, respectively class name of object and class version) |
| **Value:** | Parameter value or index of the binary objects |
| **Num Values:** | Number of values<br>(1 for single parameters or objects, >1 for arrays ) |
| **Streamer Info:** | Identifier of the binary object storing the streamer info for ROOT objects with the ROOT version in brackets |

**Comment for new parameters:**

If **new** parameters have to be inserted, a new comment will be created with the given author and description.

**Newly introduced parameters:**

This table shows all parameters in the new parameter set, which where never stored in the FINAL tables before.

**Comparison of new and old values:**

This table shows the comparison between all parameters, which are in the new set **and** which are actually valid in the specified time range.

**Parameters, not filled:**

This table shows all parameters, which are actually valid in the specified time range and which are **not** in the new set.
**These parameters are not changed and stay valid.**

If differences have been found and no error occured (for example a change of the parameter type is not allowed), there are two buttons in the "**Compare & Insert**" mode:
Click on "**Cancel**" to unlock the tables without any changes.
Press "**Insert**" to store the version permanently and release the lock.

Figure 4.21: Help page of validation preview for condition-style parameter containers

## 4.8 Geometry

The WebDB folder Geometry contains various public applications to display the geometry information (valid versions, tree of detector volumes and volume, media, hit definitions) and non-public applications for storage and changes.
Fig. 4.22 shows as example the result of the application "**Show geometry**" after selecting the beam time APR12 in the entry form (not shown). It shows all versions validated for real runs in this time range.

**Geometry Sets**

**History status at:** now
**Experiment:** APR12
**Date range for search:** 19-MAR-2012 00:00:00 - 13-MAY-2012 23:59:59

Show alignment

| Valid Since | Valid Until | First Run | First File | Last Run | Last File |
|---|---|---|---|---|---|
| 19-MAR-2012 00:00:00 | 10-FEB-2014 23:59:59 | 132278933 | st12080222853 | 186290184 | ri13340013624 |

| Part | Vers | Version Name | Valid Since<br>First Run<br>First File | Valid Until<br>Last Run<br>Last File | Date Create | Invalid Since | Comment |
|---|---|---|---|---|---|---|---|
| CAVE | 2 | CaveIdealApr07 | 01-MAR-2007 00:00:00<br>1620123104<br>to07078140022 | 01-JAN-4000 00:00:00<br>236944250<br>co15196091050 | 02-MAR-2007 10:47:01 | 01-JAN-4000 00:00:00 | 13340 |
| MDC | 21 | MdcIdealApr12 | 19-MAR-2012 00:00:00<br>132278933<br>st12080222853 | 01-JAN-4000 00:00:00<br>236944250<br>co15196091050 | 30-JAN-2015 12:09:10 | 01-JAN-4000 00:00:00 | 40774 |
| RICH | 26 | RichIdealApr12 | 19-MAR-2012 00:00:00<br>132278933<br>st12080222853 | 01-JAN-4000 00:00:00<br>236944250<br>co15196091050 | 15-JUN-2012 11:17:56 | 01-JAN-4000 00:00:00 | 36268 |
| RPC | 7 | RpcIdealApr2012Gen7 | 19-MAR-2012 00:00:00<br>132278933<br>st12080222853 | 01-JAN-4000 00:00:00<br>236944250<br>co15196091050 | 28-MAY-2014 14:04:16 | 01-JAN-4000 00:00:00 | 40356 |
| SECT | 19 | SectIdealApr12 | 19-MAR-2012 00:00:00<br>132278933<br>st12080222853 | 01-JAN-4000 00:00:00<br>236944250<br>co15196091050 | 15-JUN-2012 11:20:38 | 01-JAN-4000 00:00:00 | 36233 |
| SHOWER | 2 | ShowerIdealNov01 | 01-JAN-2001 00:00:00<br>987581231<br>xx01108100711 | 01-JAN-4000 00:00:00<br>236944250<br>co15196091050 | 20-JAN-2004 13:14:55 | 01-JAN-4000 00:00:00 | 5193 |
| TARGET | 62 | TargetApr12_Au_15seg_ideal | 19-MAR-2012 00:00:00<br>132278933<br>st12080222853 | 10-FEB-2014 23:59:59<br>186290184<br>ri13340013624 | 23-APR-2014 09:22:22 | 01-JAN-4000 00:00:00 | 36271 |
| TOF | 18 | TofIdealJan2010 | 01-JAN-2010 00:00:00<br>83503832<br>te10246105032 | 01-JAN-4000 00:00:00<br>236944250<br>co15196091050 | 09-JUN-2010 12:21:15 | 01-JAN-4000 00:00:00 | 28197 |
| WALL | 2 | WallIdealApr12 | 19-MAR-2012 00:00:00<br>132278933<br>st12080222853 | 01-JAN-4000 00:00:00<br>236944250<br>co15196091050 | 15-JUN-2012 11:22:42 | 01-JAN-4000 00:00:00 | 36276 |

Back to entry form

Figure 4.22: Geometry set for beam time APR12

If one clicks on the link of a version name one gets the full geometry tree of this version. Fig. 4.23 shows on the left side for example the MDC geometry tree. The link of the volume name pops up a separate window and shows the volume definition, here on the right side for the volume DR1M1, the MDC plane 1 mother volume in sector 1.

**Tree of Detector Part MDC Version: MdcIdealApr12** (Status at 01-JAN-4000 00:00:00)

```
CAVE --- SEC1 --- DR1M1 --- D1F1 --- D1I1
                        --- D1F2 --- D1I2
                        --- D1A1 --- D1G1 --- D1C2
                        --- D1A2 --- D1G2 --- D1C3
                        --- D1A3 --- D1G3 --- D1C4
                        --- D1A4 --- D1G4 --- D1C5
                        --- D1A5 --- D1G5 --- D1C6
                        --- D1A6 --- D1G6 --- D1C7
                        --- D1Z1 --- D1J1 --- D1W1
                                          --- D1C1
                        --- D1K0 --- D1V0 --- D1S1
                        --- D1K1 --- D1V1 --- D1S2
                        --- D1K2 --- D1V2 --- D1S3
                        --- D1K3 --- D1V3 --- D1S4
                        --- D1K4 --- D1V4 --- D1S5
                        --- D1K5 --- D1V5 --- D1S6
                        --- D1K6 --- D1V6 --- D1W2
                 --- DR2M1 --- D2F1 --- D2I1 --- D2W1
                        --- D2A0 --- D2L0
                                 --- D2L1 --- D2C1
```

**DR1M1**

**Medium:** AIR1
**Geant3 Shape:** TRAP
**Points:**

| | | |
|---|---|---|
| 53.629 | -385 | -27.3 |
| 451.028 | 575 | -27.3 |
| -451.028 | 575 | -27.3 |
| -53.629 | -385 | -27.3 |
| 74.152 | -385 | 25.376 |
| 471.561 | 575 | 25.376 |
| -471.561 | 575 | 25.376 |
| -74.152 | -385 | 25.376 |

**Mother Volume:** SEC1
**Rotation Matrix:**

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | .716965 | .697109 |
| 0 | -.697109 | .716965 |

**Position:** 0 392.354 403.529
**Reference Coordinate System:** CAVE
**Min Geometry Version:** 21
**Max Geometry Version:** 21
**Created:** 30-JAN-2015 10:40:31

Figure 4.23: Mdc geometry tree of version MdcIdealApr12

If one would instead select the simulation project APR12 in the entry form of the application one would see two sets for the two reference runs apr12sim_mediumfieldideal_auau with ideal geometry and apr12sim_mediumfieldalign_auau with the alignment geometry. Each set also includes the valid versions of the coils, the frames and the Start detector (not needed for the APR12 analysis). Instead of the link **Show alignment** one would see the link **Show hit definition**.

Fig. 4.24 shows the corresponding alignment versions for beam time APR12.

**Versions of Alignment**

**Context:** GeomProduction - Beam
**Valid At:** 13-MAY-2012 23:59:59
**Status At:** 29-JUL-2015 15:52:22

| Detector Id | Version | Valid Since | Valid Until | Created At | Invalid Since | Comment |
|---|---|---|---|---|---|---|
| 1 | 161 | 19-MAR-2012 00:00:00 | 13-MAY-2012 23:59:59 | 16-SEP-2014 16:25:21 | 01-JAN-4000 00:00:00 | 40718 |
| 2 | 143 | 19-MAR-2012 00:00:00 | 31-MAY-2014 23:59:59 | 04-JUL-2014 17:43:37 | 01-JAN-4000 00:00:00 | 35613 |
| 5 | 141 | 19-MAR-2012 00:00:00 | 31-MAY-2014 23:59:59 | 08-JUL-2014 17:36:26 | 01-JAN-4000 00:00:00 | 35589 |
| 10 | 145 | 19-MAR-2012 00:00:00 | 10-FEB-2014 23:59:59 | 23-APR-2014 09:24:04 | 01-JAN-4000 00:00:00 | 37165 |
| 13 | 140 | 19-MAR-2012 00:00:00 | 31-MAY-2014 23:59:59 | 08-JUL-2014 17:35:25 | 01-JAN-4000 00:00:00 | 35594 |

**Number of Detector Ids:** 5
**Valid for date range:** 19-MAR-2012 00:00:00 - 13-MAY-2012 23:59:59
**Last update:** 16-SEP-2014 16:25:21

Figure 4.24: Alignment for beam time APR12

Unfortunately the generic interface does not allow to display the detector by name, but only by id:
1: TOF, 2: MDC, 5: Shower, 8: Start, 10: Target, 11: Forward Wall, 13: RPC, 14: EMC

The link of the version number shows the alignment LAB-transformation, for example in fig. 4.25 the 15 target positions of the segmented Gold target.

**Parameter Container: Alignment**
**Version: 145**

**Version Comments:**

| Original Context | Run Type | Run Id | Author | Description | Created |
|---|---|---|---|---|---|
| GeomProduction | Beam | 133444082 | O. Pechenova | Target Alignment Parameters for Apr12 | 27-NOV-2012 10:38:34 |

**Detector Id:**
10

**Detector Id: 10**

| detector | geomObjId | objName | r11 | r12 | r13 | r21 | r22 | r23 | r31 | r32 | r33 | px | py | pz |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TARGET | 4677 | TARG1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -54.432 |
| TARGET | 33410 | TARG10 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -22.492 |
| TARGET | 33415 | TARG11 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -18.885 |
| TARGET | 33420 | TARG12 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -15.404 |
| TARGET | 33492 | TARG13 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -11.737 |
| TARGET | 33497 | TARG14 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -8.148 |
| TARGET | 33502 | TARG15 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -4.492 |
| TARGET | 4695 | TARG2 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -51.503 |
| TARGET | 6552 | TARG3 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -47.569 |
| TARGET | 10298 | TARG4 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -44.324 |
| TARGET | 33385 | TARG5 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -40.437 |
| TARGET | 33390 | TARG6 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -37.185 |
| TARGET | 33395 | TARG7 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -33.199 |
| TARGET | 33400 | TARG8 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -30.193 |
| TARGET | 33405 | TARG9 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | .025 | .78 | -26.433 |

Rows: 1 - 15

Figure 4.25: Target alignment for beam time APR12

## 4.9 Explore the WebDB Site

Take some time and click on some folders and applications. Most pages accessible by user HADES (and all other accounts besides HADES_ANA) have "Help" pages.

Table 4.1 shows these not-restricted folders and a short description.

| *Main folder* | *Description* |
|---|---|
| Beam time & runs | Experiment infos, DAQ runs and logbooks |
| DAQ & TRB | DAQ and Trbnet configuration, TRB2 INL corrections |
| Magnet and Cryo | Monitoring of Magnet, Cryo and Cave temperatures and pressures |
| Slow Control | Root folder for interfaces to Slow Control data (contains sub-folders for online and offline storage) |
| RICH | Folder for RICH detector since 2010 (contains sub-folder for data before 2010) |
| MDC | Folder for MDC detector since 2010 (contains sub-folder for data before 2010) |
| RPC | Folder for RPC detector since 2010 (contains sub-folder for data before 2010) |
| Analysis ⇒ Hydra1 Params | Hydra1 parameters with sub-folders for condition- and tree-style parameter containers |
| Analysis ⇒ Hydra2 Params | Hydra2 parameters with sub-folders for condition- and tree-style parameter containers |
| Analysis ⇒ Simulation | Simulation projects and runs |
| Analysis ⇒ Geometry | Geometry for simulation and analysis |
| Analysis ⇒ Run Validation | Run validation for DST production |
| Analysis ⇒ Analysis logbook | old HADES Analysis Logbook not used anymore |
| HADES Collaboration | HADES institutes, people and author lists |
| Documentation | Documentation of Oracle accounts and utility software |

Table 4.1: WebDB folders accessible by user HADES