Die HODO Library

The software for the hodoscopes is capsulted in **libHodo** library. The **HHodoDetector** is defined for ten modules with 128 channels each. As the hodoscopes are read out by multihit TDCs, the software is able to store several hits per channel per event. The exact number depends on the level of analysis.

The fact that now different detector types are using the hodoscope class introduces additional complexity. Different detector types need for example different hitfinders. In this case the hitfinder (or other task) is not running on all hodoscope modules, but it is limited to only one module (remark: In the future this could also be done by adding a new parameter container which stores that information). At the moment this is done by adding an option line to the *HHodoTaskSet::make()* function which tells the taskset the detector types. The exact syntax might change, therefore check the sourcecode details.

Classes which are common to all detectors and/or not especially relevant for **HHodo**, like parameter-i/o, will not be listed here.

Important dependencies:

HHodoHit2Start and **HHodoUnpacker** depend on "start" package (for obvious reasons). **HHodoTrbUnpacker** is based on **HTrbBaseUnpacker** contained in the "base" library.

Data storage classes

The hodoscope data is stored in the following classes:

- **HHodoRaw -** Contains up to four hits with time and width per channel. **HHodoRaw** is filled by **HHodoTrbUnpacker** or **HHodoUnpacker**. There are two ways of filling this category, *fill(time,width)* (as used by **HHodoTrbUnpacker**) and *fill_lead(time)* / *fill_trail(time)* (used by **HHodoUnpacker**). The second one calculates the width by itself and takes care about hits without leading or trailing time. Hits which have only a trailing time are discared! The fill functions take care about hits which exceed the maximum number of storable hits. In such a case the multiplicity will be further increased, but not time/width will be stored. The return value of the fill function will be *false* (this does not imply an error!). In any case, no overflow checking has to be done by the unpacker.
  Reading data from **HHodoRaw** is done by calling *getTime(n)* / *getWidth(n)* where n is the number of hit to return, starting at one. An over or underflow will return an error. The caller is responsible to check that n is inside range, for instance by checking against the *getMaxMult()* return value. The time and width of an unfilled hit is alway -1. The number of hits can be asked for by *getNHits()*, which might be

larger than the maximum multiplicity of four!
Please do not check against number four as this number could (but most likely will never) be increased. Check always the function *getMaxMult()* instead.

- **HHodoCal** - Containing Time and ADC, up to four, same as on raw Level. Note: Calibrator will not fill for than four hits, thus number of hit will (at least at the moment) differ from raw level and not exceed four. The behaviour is nearly the same as the raw level category, there are only small differences in the function names. The fill function is replaced by *setTimeAdc()* and width is always adc after calibration. The time and width of an unfilled hit is alway -10^6 and -1 accordingly.

- **HHodoHit** - Until now only one hit per module is supported! Reason: Events with more than one hit in start detector will be discarted anyway. In the future this should be changed (especially for pion beam).

Unpacker tasks

- **HHodoTrbUnpacker** - This is the default unpacker for hodoscope data. It reads TRB data (board/subevent id has to be given by parameter) and fills the **HHodoRaw** category. This tasks is derived from **HTrbBaseUnpacker** where all the real unpacking and error handling takes place. Using **TrbLookup**. The time correction is done based on the TRB software version if not disabled by *disableTimeRef()*. In addition this task is creating (if enabled by *enableControlHistograms()*) diagnosis plots. On these plots the TRB windows can be checked as well as the software windows set in **HodoRefHitPar**. The first plot ("TRBDiagRaw_id") contains the leading time for all channels of the TRB before time reference substraction (do not mix this up with the trigger substraction inside TRB!). In channel second part of the histogramm the same channels appear under the condition that the leading time was inside the raw time cuts defined by **HodoRefHitPar**. Note: on the left side all active channels of the TRB appear, on the right side only the ones which are connected to Hodoscopes. In the second plot ("TRBDiagDif_id") the leading time is plotted after the time reference substraction under the same conditions as in the first plot. Both plots are saved to postscript files and to the unpacked root dst file.
  The **HTrbBaseUnpacker** is able to stored 10 hits per channel, the **HHodoTrbUnpacker** will fill all hits inside the given time window inside **HHodoRaw** which will accept all, but only store the first four hits.
  Even so the time reference is substracted, the raw time can never be negative, because there is always a large positive number added (40000 corresponding to $4\mu s$, which is bigger than any useful time window in the TRB).

- **HHodoUnpacker** is unpacking data from the standard CAEN 32 channel TDC plus the CAEN v1190 multihit TDC. It is only used in cases where on of these VME modules is used in the TOF/Trigger/Scaler crate (compatibility with files

taken before 2006). The Unpacker works on subevent with id 416 and is using the **HodoLookup** table. There should be no reason to use this task again as in the present and future all hodoscope data is read out by the TRBs.

The following tasks are connecting the storage classes:

- **HHodoCalibrator** is reading data from **HHodoRaw**, doing calibration using **HodoCalPar** and then storing the data again in **HHodoCal**. The calibration for the time is using a slope and an offset parameter $t_{new} = t_{old} \cdot \text{slope} + \text{offset}$. For the adc (which is the width of the signal) parameter containers are existing, but they are not used. The corresponding code is commented out, the width is copied without any processing, because up to now the adc values are neither used nor is there a way of creating these parameters. As there is no way to calibrate non existing data, the maximum hit number is four, higher multiplicieties are discarted.

- **HHodoHitFDiamond** is the hitfinder for diamond detectors connected to TRBs. This hitfinder is not a copy from the Start detector class! It is only looking for hits in one strip, without any fancy timing or amplitude calculation if more than one strip was hit. If more than one stripe was hit no hit is generated. Because of the simplicity it can also be used for the old scintillating bar hodoscopes, where the probability of a particle hitting to stripes is practically zero. The preformance is not very good as this task was only build as a quick solution for the new diamond detectors read out by the TRBs. The hitfinder is using **HHodoCal** data and is applying the cut defined in **HodoRefHitPar** before checking the multiplicity of hit stripes. On which of the hodoscope modules this task is working has to be defined by *set_ mod(module)*.

- **HHodoHitFFiber** is the hitfinder for the scintillating fiber hodoscopes, wher the probability that are particle is fireing more than one fiber is ~60%. The task will generate hits even if the multiplicity inside the time windows is higher than one. At the moment the alorithm check that is quiet simple: Only mulitiplicities up to three, where all fibers are neighbours, are taking into account. No energy checks or any other fancy pattern search is done. The preformance could be significantly improved by that. The hitfinder is using **HHodoCal** data and is applying the cut defined in **HodoRefHitPar** before checking the multiplicity of hit stripes. On which of the hodoscope modules this task is working has to be defined by *set_ mod(module)*.

- **HHodoHit2Start** - This task copies hit data from hodo hit to start detector hit category. This is needed for compatibility with software which relies on diamond start detector signal in the **HStartHit** category, for example TOF and TOFino code. As we did not want to change these parts of software this workaround task was implemented. On which of the hodoscope modules this task is working on has to be defined by *set_ mod(module)*.

Parameter tables

Following is the list of parameter containers used by the Hodo library. There are examples for every parameter.

- **HodoLookup** is the lookup table which is only used by the (depreciated) HHodoUnpacker. It is only needed if TDC data from the "TOF/Trigger/Scaler Crate" has to be imported into a Hodo detector module.

```
# Lookup table for the Hodo unpacker
# Format:
# crate slot channel module strip
[HodoLookup]
3 1 0 1 0
```

- **TrbLookup** is the lookup table for all TRB boards. It is connecting board ID and channels with detectoty type and corresponding module and channel. Attention: **TrbLookup** is shared by Hodo, Forward Wall and RPC detectors, which share the same table. For an update of the table into the database this has to be taken into account!

```
# Lookup table for the TRB unpacker
# Format:
# subeventId TrbChannel detector sector module cell side
# detectors: T-TOF, F-Tofino, S-Start, H-Hodo, W-ForwardWall, R-RPC)
# side: l-left, r-right, m-middle, u-up, d-down
[TrbLookup]
806 64 H 0 1 80 m
```

- **HodoRefHitPar** contains additional parameters which are used by the HHodoTrbUnpacker and the HitFinders. The first two parameters are channel windows for the unpacker, signals outside these windows are not taken into account (in channel numbers, normally 100ps). The second pair is a time cut for the hitfinders (in ns).

```
# Reference Hits for Hodo
# Format:
# Modul UnpLow UnpHigh HitfLow HitfHigh
[HodoRefHitPar]
0 30000 40000 32500 32600
```

- **HodoCalPar** stores the calibration parameters for the hodoscopes. For every channel there are two pairs of numbers. The first one define the slope and offset for the time calibration (slope should be 0.1 for getting from 100ps channel width to 1ns; offset is in ns after slope correction). The second pair is not in use yet. The ADC

data is at the moment not touched by the calibrator. The code is existing, but its is commented out.

```
# Hodo Calibration parameters
# Format:
# module strip time-slope time-offset adc-slope adc-offset
[HodoCalPar]
1 2 0.1 0.0 1.0 0.0
```

Example code

Example source code for unpacking, calibrating and hitfinding. The pseudo code contains only the for the hodoscope stuff necessary lines, all common initialization has been left out.

```
//------ define detector ------
Int_t nHodoMods[4] = {1,1,0,0}; // active modules
spec->addDetector(new HHodoDetector);
spec->getDetector("Hodo")->setModules(-1,nHodoMods);
//------ add unpacker for TRB 801 ------
HHodoTrbUnpacker *tp;
tp=new HHodoTrbUnpacker(801);
tp->enableControlHistograms();
source->addUnpacker(tp);
//------ now calibrate data ------
taskSet->add( new HHodoCalibrater("hodo.cal","Hodo cal") );
//------ hitfinder for module 0
HHodoHitFFiber *t;
t=new HHodoHitFFiber("hodo.hitfF","Hodo 0 hitf");
t->set_mod(0);
taskSet->add( t);
//------ hitfinder for module 1 ------
HHodoHitFDiamond *d;
d=new HHodoHitFDiamond("hodo.hitfD","Hodo 1 hitf");
d->set_mod(1);
taskSet->add( d);
//------ copy module 1 to start hit category ------
HHodoHit2Start *h2s;
h2s=new HHodoHit2Start("hodo.hit2start","Hodo Hit2Start");
h2s->set_mod(1);
taskSet->add( h2s);
```

Remark:

Calibration and hitfinding can also be done with the HodoTaskSet by calling *make("","real,fiber=0,dian*
with the apropriate option string.