New VETRA v6r0

Default configuration

The latest edition of the Vetra project consists of NZS data analysis for both Velo and ST. This Twiki is devoted to the Velo part alone.

There are availabe a lot of data that came from different testbeam studies (acdc2 and acdc3), lab tests, slice tests, commissioning, etc. This makes it very difficult to provide one generic options files set that would be used for all of the data types. It was decided that for the default Vetra job, available out of the box, the input data file will be one of the generated test files used for validation of the emulation chain. The generated files used as default input are stored at *CASTOR*. User can easily configure any Vetra job using default options as a start point.

--> Main options file - VetraTELL1NZS.opts

Structure of the main option file:

1) General settings

```
//=============================================================================
// General settings
#include "$STDOPTS/LHCbApplication.opts"
#include "$STDOPTS/RawDataIO.opts"
#include "$STDOPTS/DigiDicts.opts"
// Monitoring - histogram persistency Svc
#include "$STDOPTS/RootHist.opts"
RootHistCnv.ForceAlphaIds = true;
HistogramPersistencySvc.OutputFile = "NonZeroSuppressedData.root";
//#pragma print off
/// time profiling
AuditorSvc.Auditors = { "ChronoAuditor" };
ApplicationMgr.ExtSvc += { "AuditorSvc" };
ChronoAuditor.Enable = true;
///
//=============================================================================
```

In this section some useful and generic for all LHCb projects options are included. The one of most importance form the NZS data analysis point of view is the file RawDataIO.opts that makes possible to read raw data files in *mdf* format.

2) Database access.

This part is somewhat involved - mainly becaus of multiple data that can be used.

2.1) Testbeam (acdc3) and tracking. In order to run a Vetra job using data files taken during the acdc3 testbeam one needs to use the test beam geometry description. This will allow to run not only the emulation part but also to perform standalone Velo tracking. To run over the acdc3 data one needs to include:

```
MagneticFieldSvc.FieldMapFile = "$FIELDMAPROOT/cdf/field047.cdf";
/// --> uncomment this if you want to use ACDC DetDsc <--
DetectorDataSvc.DetDbLocation = "$XMLDDDBROOT/DDDB/Velo/VeloACDC/ACDC3.xml";
```

The testbeam geometry is stored in xml files and is a part of the Velo/VeloACDC package that by default is a part of Vetra project.

2.2) Testbeam emulation.

It is also possible to run emulation part over the testbeam data without tracking part using dynamically created Velo partition of the condition data base (the condition database is described later on in this document). To do so one needs to use the following:

```
#include "$DDDBROOT/options/DC06.opts"
MagneticFieldSvc.FieldMapFile = "$FIELDMAPROOT/cdf/field047.cdf";
/// --> uncomment to have quasi test beam setup with Cond DB <--
#include "$VETRAROOT/options/Velo/VeloFakeHP4.opts"
#include "$VETRAROOT/options/Velo/Tell1CondHP4.opts"
//--
```

The VeloFakeHP4.opts are used in order to connect to the SQLite file where processing parameters needed by the TELL1 algorithms are stored. In this configuration default LHCb conditions database will also be used. The LHCb partition is needed to get for instance information about sensors. Mapping between TELL1 boards and software sensor numbers are probvided via Tell1CondHP4.opts and is compatible with the acdc3 HP4 cable configuration (other testbeam cable configurations can be created if needed).

2.3) Full Velo setup - commissioning.

So far the commissioning exercise has been performed using static configuration of the processing algorithms only. By static configuration we mean taking processing parametrs from the text files rather than from the database. In case of static configuration each TELL1 board is treated in the same way - no differences are taken into account (for example dead strips, noisy channels etc). Beside static mode it is also possible to run data processing in dynamic mode - using database to configure emulation algorithms. Including the following options allows to run Vetra job using SQLite database:

```
/// --> uncomment this if you want to use Velo Cond partition <--
#include "$VETRAROOT/options/Velo/VeloCondDB.opts"
```

Appropriate mapping file is also needed - look at *commissionig* branch of the *options/Velo*.

2.4) Other - depends on what you want to do - in most cases it is enough to use existing examples - or contact an expert

3) Input data section.

3.1) MDF file stored locally on the disk:

To connect to a data file that is available locally one needs to use following 'magic' line:

```
//-----> Main event input <-----//
EventSelector.Input += {
  "DATAFILE='file:///SOME_PATH/my_favorit_raw_data_file.mdf' SVC='LHCb::MDFSelector'"
};
```

3.2) MDF file store on the CASTOR

The appropriat line is as follows:

```
//-----> Main event input <-----//
EventSelector.Input += {
  "DATAFILE='rfio:/castor/cern.ch/user/s/szumlat/Vetra_v6r0_TestData/clusterGeneration_phi_070525
};
```

This file is real and can be actually used to test the connection.

4) Vetra processing sequences

Definition of the processing chain is following:

```
//--------------------------
// Vetra sequences - analysis of Non-zero suppressed real data
ApplicationMgr.TopAlg += {
                           "ProcessPhase/TELL1Processing"
                          ,"ProcessPhase/Moni"
                         };
TELL1Processing.DetectorList+={ "VELO" };
TELL1ProcessingVELOSeq.Members+={ "TELL1BinaryOutputChecker" };
TELL1ProcessingVELOSeq.Members+={ "PrepareVeloFullRawBuffer" };
TELL1ProcessingVELOSeq.Members+={ "DecodeVeloFullRawBuffer" };
// --> This may be relevant <--
//PrepareVeloFullRawBuffer.RunWithODIN=false;
// --> -------------------- <--
```

The TELL1BinaryOutputChecker is a simple but useful algorithm for checking if a mdf data file has been properly built - it also allows to take a look at the file content by using option:

```
TELL1BinaryOutputChecker.PrintBankInfo=false;
```

This algorithm requires the following option to be set properly:

```
TELL1BinaryOutputChecker.ExpectedNumberOfTELL1s=1;
```

Also, incomplete events - missing banks - can be discarded by setting:

```
TELL1BinaryOutputChecker.DropIncompleteEvents=true;
```

If data file used has been built wihtout ODIN bank - testbeam, lab test, etc, the two following options must be included:

```
PrepareVeloFullRawBuffer.RunWithODIN=false;
TELL1BinaryOutputChecker.RunWithODIN=false;
```

4.1) Processing sequence.

This sequence consists of two task - raw data decoding and subequent emulation of the decoded data. Decoding is performed by two algorithms:

```
TELL1ProcessingVELOSeq.Members+={ "PrepareVeloFullRawBuffer" };
TELL1ProcessingVELOSeq.Members+={ "DecodeVeloFullRawBuffer" };
```

This part of job does not require any interaction from a user apart from setting RunWithODIN option (see above point 4) if necessary.

4.2) Emulation

This part of the emulation performs actuall zero-sruppression of the NZS data and produces standard raw bank that contains VeloClusters object. Actual sequence of algorithms is defined in the options file:

```
/// --> TELL1 emulation
#include "$VETRAROOT/options/Velo/TELL1Emulator.opts"
///
```

The definition if given below:

```
TELL1ProcessingVELOSeq.Members += {
                              "VeloTELL1EmulatorInit"
```

```
 ,"dataTranslator"
 ,"VeloTELL1PedestalSubtractor"
 ,"VeloTELL1FIRFilter"
 ,"VeloTELL1MCMS"
 ,"VeloTELL1Reordering"
 ,"VeloTELL1LCMS"
 ,"VeloTELL1ClusterMaker"
};
```

There are some crucial things to be set in order to perform emulation sequence accordingly.

ConvergenceLimit - this is a number of event needed to train the pedestal following procedure. To get rid of any instabilities this should be set to 4000:

```
VeloTELL1EmulatorInit.ConvergenceLimit=4000;
```

In case of lab test data this should be set to 1 since the pedestal value is fixed for this kind of studies.

ProcessEnable/ForceEnable - this options allow to override the enable flag set by decoding. Regular configuration of the emulation procedure is performed dynamically using input data file (header) to set active list of processing. However, sometimes we want to override this - mainly beacuse development study. This can be done by via ForceEnable flag, for example following options allow to run MCMS algorithm:

```
VeloTELL1MCMS.MCMSProcessEnable=1;
VeloTELL1MCMS.ForceEnable=true;
```

STATIC/DYNAMIC configuration of the emulator.

By default the emulator is configured in STATIC mode and configuration of the processing algorithms is done via options files. This requires no input from user. In case of running emulation in DYNAMIC mode one needs to add this line:

```
/// --> include when run in DYNAMIC mode
#include "$VETRAROOT/options/Velo/TELL1EmulatorCondDB.opts"
```

List of TELL1 boards - the list of used TELL1 board is required in order to configure properly processing algorithms when running in DYNAMIC mode. In the next release this will be a part of the database and providing TELL1 list via options file will be obsolete.

5) Monitoring - basic monitoring of the raw data.

In this section only the basic monitoring of the raw data is described. The monitorig is performed at each stage of the TELL1 processing. Also, decoded raw ADC data can be monitored. The monitoring sequence can be swith on by including options:

```
/// --> for experts raw samples display
#include "$VETRAROOT/options/Velo/TELL1Checkers.opts"
```

6) Output data stream.

The main output of Vetra job is standard LHCb raw bank of type 8 (Velo). This bank can be saved in a file by using:

```
#include "$VETRAROOT/options/Velo/VetraDigiWriter.opts"
```

The file can be subsequently analysed (to compare emulated raw bank with real one - produced by the TELL1 board) or use as the input for reconstruction (Brunel with Velo stand alone tracking).

Condition Data Base

In the new release parameters for the processing algorithms can be defined in either **STATIC** or **DYNAMIC** modes. This part of the description will focuse on the DYNAMIC configuration and explain how to create the condition Data Base for the Vetra.

In the current version one needs to use two scripts to create a SQLite file with conditions. The steps are described below

1. Creation of the valid XML description of the conditions.

In order to create the file it takes to run a python script ***write_velo_xml_cond.py***. The script is located in the *Vetra/v5r2/python* directory. The file created by the script - ***VeloTELL1Cond.xml*** - will be placed in the *Vetra/v5r2/VetraCondDB/Velo/VeloCondDB* directory by default (the default location should not be changed by user at any rate!). VeloTELL1Cond contains a collection of TELL1 conditions that in turn contain processing parameters needed for both the emulation and the TELL1 boards configuration. The example of a condition containing just a single number:

```
<condition classID="5" name="VeloTELL1Board0">
  <param name="pedestal_enable" type="int">
   1
  </param>
```

and a vector of numbers:

```
<paramVector name="pedestal_sum" type="int">
   524288 524288 524288 524288 524288 524288 524288 ....
</param>
```

In order to create the data base file properly another xml file is needed that specifies which conditions are to be put to the data base. The file is located in the same directory as VeloTELL1Cond.xml file and is called velocond.xml. A snipet of the xml code from the file is presented below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DDDB SYSTEM "../DTD/structure.dtd">
 <DDDB>
   <catalog name="VeloCondDB">
     <!-- R sensors - 0:41 -->
     <conditionref href="VeloTELL1Cond.xml#VeloTELL1Board0"/>
     <conditionref href="VeloTELL1Cond.xml#VeloTELL1Board1"/>
     ...
 </DDDB>
```

Each condition we want to have in the Condition Data Base should be declared there.

2. Writing xml condition into the SQLite file.

To create a valid SQLite data base file one needs to run a script called ***create_sqlite_file_from_xml.(c)sh*** located in *Vetra/v5r2/scripts/* directory (just by executing source from the Vetra *cmt/* directory). The script will search for the file with defined entry point to the data base (in our case the entry point is defined in the *lhcb.xml* file that can be found in the *Vetra/v5r2/VetraCondDB/Velo/* directory. The most important part of the file from the Vetra point of view is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DDDB SYSTEM "DTD/structure.dtd">
<DDDB>
  <!--  *******************************************************  -->
```

```
  <!--  *           LHCb Detector Description Data Base          *  -->
  <!--  *              new hierarchy detector setup              *  -->
  <!--  ********************************************************  -->
  <catalog name="dd">
 ...
 <catalogref href="VeloCondDB/velocond.xml#VeloCondDB" />
</DDDB>
```

This link points to the file where Vetra conditions are specified. The *create_sqlite_file_from_xml.(c)sh* will create the SQLite data base file at */Vetra/v5r2/VetraCondDB/Velo/* with the default **VELOCOND.db** name for the file and **VELOCOND** for the data base name. At this point the data base with the base line procesing parameters is ready to use within the Vetra!

Checkings

After the SQLite file has been produced it is worth to make sure that the file contains everything we need. The most obvious test is to just look inside the file using ***CondDBBrowser.py***. To do so, one can perform as follow:

1. Setup the Vetera v5r2 environment (cmt config and source setup.(c)sh) 2. This is a kind of magoc line to have all you need to run graphical UI to the ConditionDB

SetupProject LHCb v23r2 --use=Tools/CondDBUI

3. Open the GUI by executing:

CondDBBrowser.py

4. Connect to the data base providing location of the data base and its name (for the described default settings the location is *Vetra/v5r2/VetraCondDB/Velo/VELOCOND.db* and the DB name is *VELOCOND*)

5. You are ready to take a peek inside the DB

Using the data base from Gaudi job

There are a few magic lines that need to be add to the option file to use our new and shine CondDB. **Note** the Vetra DB will be added as a layer on the top of default LHCb data bases. Should there be any need for some information that are not in the VELOCOND.db the appropriate service will peer through the Vetra DB and start to search the remaining DBs. This of course needs to be passed to the framework by appropriete options. The full description and examples are given below.

1. By the default this line should be added to the option file:

```
#include "$DDDBROOT/options/DC06.opts"
```

2. Next we need to create an appropriate service to read our DB:

```
CondDBCnvSvc.CondDBReader="CondDBLayeringSvc";
```

3. Let create layers of the DB we will be using (first one will be the Vetra CondDB and the LHCb DBs beneth it)

```
CondDBLayeringSvc.Layers={'CondDBAccessSvc/VELOCOND', 'CondDBDispatcherSvc'};
```

4. The last point is to provide approprate connection string:

```
VELOCOND.ConnectionString="sqlite_file:$VETRAROOT/VetraCondDB/Velo/VELOCOND.db/VELOCOND";
```

Now the Vetra CondDB will be visible from inside a Gaudi job.

-- TomaszSzumlak - 12 May 2008

This topic: LHCb > VetraHowTo
Topic revision: r4 - 2008-06-04 - TomaszSzumlak