This is the help page for ST Vetra Analysis for version **v3r1** and before. For latest version please read LHCbSTVetraAnalysis.

If you want to know how to build a ST Vetra conditions database from scratch and how make it useful for TELL1 configuration, have a look at the STVetraCondHowTo.

This package is designed to determine the ST TELL1s thresholds, pedestal values and masks and the disabled beetles and beetle ports. Scripts are provided to handle the different outputs and write the obtained values in the database(s). Finally, a set of python scripts allow to compare the values in different databases.

Here is the list of required packages (be sure you're using the head version) :

- **ST/STVetraAnalysis**
- **ST/STTELL1Event**
- **Det/DetDesc**
- **Det/DetCond**
- **Det/STDet**
- **ST/STKernel**
- **Tell1/VetraKernel**

STVetraAnalysis contains the following algorithms :

- Noise extraction and threshold setting
- First order header correction (the wanted strip is corrected according to the last header value)
- Second order correction (the wanted strip is corrected according to the two last header values)
- Noise cluster rate.
- Bad strip masking (according to readout errors and data analysis, two algorithms).
- Pedestal extraction and formatting.
- Noise per sector computation (produce xml that goes into DDDB).

The package contains also python scripts, which allow you to :

- the produced parameters (thresholds, header correction values) in the CondDB,
- create a dummy DB-like xml file,
- merge the two bad strip masks,
- dump parameters from DB into config files.
- extract parameters from xml files.
- compare a given set of databases (DDDB, LHCBCOND, Vetra CondDB).

Finally, some perl scripts are included in the package as well :

- one allows you to dump the output of STNoiseMonitor into DDDB
- one that makes comparison between any pair of xml files
- one to configure the DB comparison

I added an 'options' directory, containing a default options file, which is copied and modified when you are comparing DBs. The directory contains also a Config.xml file, used to produce the data structure and the TTrees used to compare DBs.

In order to run any Vetra Analysis algorithm, the data have to be decoded first, using STFullDecoding (ST/STOnline). You'll also need a condition DataBase, follow the basic instruction given by Tomasz here or Anne here. If you want to get thresholds and cluster rate, the Emulator has to be run as well.

If you want to run on ZS data, you'll have to use STErrorDecoding (ST/STDAQ) instead of STFullDecoding.

ST condition database

First, you need to create the xml files, using write_st_xml_cond for both IT and TT. Then source the two following bash scripts : create_it_sqlite_file_from_xml.sh and create_tt_sqlite_file_from_xml.sh. Now your database is ready. Note that the condition path you have to give to the Emulator is just "CondDB". For intance, if you want to run on IT data, the DB-related options are :

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

CondDBCnvSvc.CondDBReader = "CondDBLayeringSvc";
CondDBLayeringSvc.Layers = {"CondDBAccessSvc/COND",
                            "CondDBDispatcherSvc"};

COND.ConnectionString = "sqlite_file:$VETRAROOT/VetraCondDB/IT/COND.db/COND";
```

All you have to change when running on TT data is the connection string (replace "IT" with "TT").

The Algorithms

/

The two algorithms work in the same way, they compute a correction (for any strip you want) according to the header value(s). The needed sequence is :

```
ApplicationMgr.TopAlg = {"STFullDecoding", "STFirstCorrection"};

STFirstCorrection.DetType = "IT";
STFirstCorrection.ConditionLocation = "CondDB/TELL1Board";
STFirstCorrection.ProduceFile = true;
STFirstCorrection.OutputFileName = "~/HeaderCorrectionValues.txt"
```

The computed header correction values will be put (in a Condition DB-like format) in the file 'HeaderCorrectionValues.txt'.

As an output you'll get the CondDB-like file and some root histograms, used to compute the corrections.

Now the second order correction is implemented in the Emulator, but we can still use the first order one. I simply put twice the same correction value, ie the correction doesn't depend on the first header bit value.

This algorithm computes the noise (per strip) in any TELL1 container (Raw/ITFull, Raw/TT/PedSubADCs, ...). The Emulator needs to be run.

```
ApplicationMgr.TopAlg =  { "STFullDecoding", "ProcessPhase/Emulator", "STThresholdFinder" };

Emulator.DetectorList += { "ST" };
EmulatorSTSeq.Members += { "STTELL1PedestalSubtractor", "STTELL1LCMS"};

STThresholdFinder.DetType = "IT";
STThresholdFinder.InputLocation = "Raw/IT/LCMSADCs";
STThresholdFinder.OutputFileName = "~/Thresholds";
STThresholdFinder.ThresholdsAndValues = {"Hit_threshold", "4.", "3072",
                                         "Confirmation_threshold", "4.5", "48"};
STThresholdFinder.ProduceFile = true;
```

The thresholds will be put in the files 'ThresholdsHit4.' and 'ThresholdsCon4.5' (in a CondDB-like format), and some root histograms will be created as well.

NB : depending on the InputLocation, you won't need to use the full Emulator sequence, for instance, if you want the noise in "Raw/IT/PedSubADCs", the STTELL1LCMS is useless.

STScanner

This algorithm counts the number of found clusters and gives the cluster rate (based on the EventInfo, may thus be wrong if the PCN is bugged).

```
ApplicationMgr.TopAlg = { "STFullDecoding", "ProcessPhase/Emulator", "ProcessPhase/Reco", "STScan

Emulator.DetectorList += { "ST" };
EmulatorSTSeq.Members += { "STTELL1PedestalSubtractor", "STTELL1LCMS", "STTELL1ClusterMaker" };

Reco.DetectorList+={"TT"};
RecoTTSeq.Members += {"RawBankToSTClusterAlg/Clustering"};
Clustering.detType            = "TT";
Clustering.rawEventLocation   = "Emu/RawEvent";
Clustering.clusterLocation    = "Emu/TT/Clusters";
Clustering.readoutTool        = "TTReadoutTool";


Scan.DetType = "TT";
```

The cluster rate and number are printed. Histograms are produced :

- one containing the number of cluster per event (1D histo, called clusternbr)
- one containing the size of each found cluster (1D histo, called clustersize)
- one containing the number of n-strip clusters per TELL1 board (2D histo, called strip)
- one per TELL1, containing the number of n-strip cluster per channel (2D histo, called ChanXY)

This algorithm looks at the Event Info and determines which strip have to be masked because of :

- Corrupted error bank
- Optical link disabled (is it a real error ?)
- Link loss
- No clock
- RAM full
- Event size
- No event
- Pseudo header error
- PCN error

It writes a nice report at the end of the run and put the obtained mask in DB-like format. The following sequence will produce a Error.txt file containing the mask, running on NZS data :

```
ApplicationMgr.TopAlg = { "STFullDecoding", "STErrorMasker/Error" };

STFullDecoding.DetType           = "IT";
STFullDecoding.EventInfoLocation = "Raw/IT/EventInfo";
STFullDecoding.OutputLocation    = "Raw/IT/Full";

Error.DetType                       = "IT";
Error.InputLocation                 = "Raw/IT/Full";
Error.OutputFileName                = "Error.txt";
```

If you want to use this algorithm on ZS data, the sequence becomes, for TT :

```
ApplicationMgr.TopAlg =  { "STErrorDecoding", "STErrorMasker/Dead" };
```

```
STErrorDecoding.DetType          = "TT";
STErrorDecoding.OutputLocation   = "Raw/TT/ErrorTELL1";
STErrorDecoding.BankType         = "TTError";

Dead.DetType                       = "TT";
Dead.OutputFileName                = "Error.txt";
Dead.RunOnNZS                      = false;
```

This algorithm looks at raw data and determines if a given strip is bad because of :

- low gain
- high noise
- short

I divided the low gain errors into 2 categories : there can be "normal" signal value with low noise, or low data value with "usual" noise. If you have better or other ideas / suggestion please tell me!

It also writes a summary in the end and put all information in a DB-like file. The next example shows how to create a BadLink.txt file containing the mask.

```
ApplicationMgr.TopAlg =  { "STFullDecoding", "STBadLinkMasker/Bad" };

STFullDecoding.DetType           = "IT";
STFullDecoding.EventInfoLocation = "Raw/IT/EventInfo";
STFullDecoding.OutputLocation    = "Raw/IT/Full";

Bad.DetType                        = "IT";
Bad.InputLocation                  = "Raw/IT/Full";
Bad.OutputFileName                 = "BadLinks.txt";
```

If a strip doesn't have any error, the algorithm checks into the Condition database (the "big" one, not the Vetra specific one) whether the strip is known to be bad. The output file contains also information on disabled beetles and beetle ports.

This algorithm dumps the data from Raw/IT/SubPeds into a text file, in DB-like format. The following sequence will produce a file peds.txt, containing the pedestal values for the whole TT detector :

```
ApplicationMgr.TopAlg =  { "STFullDecoding", "ProcessPhase/Emulator", "STPedestalWriter/Writer" }

STFullDecoding.DetType           = "TT";
STFullDecoding.EventInfoLocation = "Raw/TT/EventInfo";
STFullDecoding.OutputLocation    = "Raw/TT/Full";

Emulator.DetectorList += { "ST" };
EmulatorSTSeq.Members += { "STTELL1PedestalSubtractor" };

STTELL1PedestalSubtractor.DetType      = "TT";
STTELL1PedestalSubtractor.CondPath     = "CondDB";

Writer.DetType          = "TT";
Writer.InputLocation    = "Raw/TT/SubPeds";
Writer.OutputFileName   = "$MYRESULTS/peds.txt";
```

This algorithm computes the noise as a double value. The example sequence will produce a file "SecNoise.txt", containing the CMS noise values for TT :

```
ApplicationMgr.TopAlg =  { "STFullDecoding", "ProcessPhase/Emulator", "STNoiseMonitor/Noise" };
```

```
STFullDecoding.DetType           = "TT";
STFullDecoding.EventInfoLocation = "Raw/TT/EventInfo";
STFullDecoding.OutputLocation    = "Raw/TT/Full";

Emulator.DetectorList+={ "ST" };
EmulatorSTSeq.Members +=
  {
    "STTELL1PedestalSubtractor",
    "STTELL1LCMS"
  };

STTELL1PedestalSubtractor.DetType      = "TT";
STTELL1PedestalSubtractor.CondPath     = "CondDB";

STTELL1LCMS.DetType                    = "TT";
STTELL1LCMS.ConvergenceLimit           = 10000;
STTELL1LCMS.CondPath                   = "CondDB";

Noise.DetType                       = "TT";
Noise.WaitingTill                   = 10000;
Noise.InputLocation                 = "Raw/TT/LCMSADCs";
Noise.OutputFileName                = "SecNoise.txt";
Noise.ConditionLocation             = "CondDB/TELL1Board";
Noise.ProduceFile                   = true;
```

The python scripts

python script

To put the produced parameters in the CondDB, use this script this way :

```
./DumpParamIntoDB.py my_file_containing_param detector_type
```

Check at line 8 that

```
DBLOCATION = ""
```

is the correct path to your DB. detector_type can be either IT or TT. I added an alias definition in the requirement file, so that you can call the script from everywhere with

```
DumpParamIntoDB
```

write_st_xml_cond python script

I added a python script that creates a DB-like xml file, containing dummy values (ie all Hit thresholds are 6, etc). It can create the xml file for both IT or TT. The chosen detector as to be passed in argument :

```
./write_st_xml_cond.py IT
```

The file will be written in the default DataBase location, but the path can be changed by editing line 22

```
XmlFileName = ...
```

An alias has also been added, and you can run the script with

```
write_st_xml_cond
```

python script

As long as we have two ways of producing the "dead strip" mask, we have to create one file containing all the needed information before putting it into the DB. Assuming you have the two files Error.txt and BadLinks.txt and you want to create FinalMask.txt, all you have to do is :

```
MaskMerger Error.txt BadLinks.txt FinalMask.txt
```

The script takes into account the "Disable_links_per_beetle" and "Disable_beetles" fields, present in the output of STBadLinkMasker, and the order of the two source masks is not important.

NB : if you want the script to be able to merge the two files, they have to contain the same TELL1s. The best way to do so is to use the NZS mode for STErrorMasker, an example sequence is given here :

```
ApplicationMgr.TopAlg =  { "STFullDecoding", "STErrorMasker/Error", "STBadLinkMasker/Bad" };

STFullDecoding.DetType           = "IT";
STFullDecoding.EventInfoLocation = "Raw/IT/EventInfo";
STFullDecoding.OutputLocation    = "Raw/IT/Full";

Error.DetType                       = "IT";
Error.InputLocation                 = "Raw/IT/Full";
Error.OutputFileName                = "Error.txt";

Bad.InputLocation                  = "Raw/IT/Full";
Bad.OutputFileName                 = "BadLinks.txt";
```

python script

This script allows to extract values contained in the DB and puts them in config files. The arguments are 1) the detector type ("IT" or "TT") and the path to the first config file, which is number 0, as long as the DB doesn't know anything about TELL1 number. The config files must be in the same directory and named *AnyPatternYouWant* + **sourceID** + .cfg. Let's assume your "$HOME/ConfigFiles/" directory contains all cfg files you need, named "ST5.25TELL_0.cfg", "ST5.25TELL_1.cfg", ... and you want to get config files for IT. You'll have to run

```
ConfigFileWriter IT $HOME/ConfigFiles/ST5.25TELL_0.cfg
```

The script will produce 42 (48 in case of TT) cfg files, named "ST5.25TELL_x.new.cfg".

I added a functionnality which allows to run on cfg files using TELL1 Number instead of source IDs. The command is :

```
ConfigFileWriter IT $HOME/ConfigFiles/ST5.25TELL_01.cfg
```

notice that the first is now 1 instead of zero, and there are two digits!!

python script

The script allows to extract parameters contained in a xml-like txt file and writes one txt file per TELL1.

Let's assume we need the header correction values. First we run STSecondCorrection, producing Correction.txt. This file look like :

```
TELL1Board0
<paramVector name="Header_correction_analog_link_0" "type="int"comment="Lower and upper correctio
<paramVector name="Header_correction_analog_link_1"  "type="int"comment="Lower and upper correcti
```

```
<paramVector name="Header_correction_analog_link_2" "type="int"comment="Lower and upper correctio
<paramVector name="Header_correction_analog_link_3" ...
```

Then we run

```
ExtractValuesFromXML Correction.txt
```

and we get one file per TELL1, named StrippedTELL1_0, StrippedTELL1_1, ..., each of them containing the values, for StrippedTELL1_0 it would give :

```
-1 2 -1 0
1 0 0 -1
...
```

python script

The script will create a root file containing the comparison between all the possible pairs of databases you've given as arguments. The mandatory arguments are the tags for the database, which are not the actual tag of the DB version, but will be used to know what DB we are talking about. Dummy example are Old, New, Custom, ... You can specify the tag names with the --tag (or -t) option. Another mandatory field is the sub-detector type, specified by --sub (or -s).

The databases you can compare are DDDB, LHCBCOND and Vetra CondDB. You can choose to compare different version of only *one* slice. For instance, if you only want to compare the noise value per strip, the only needed DB is LHCBCOND, the script will let the default DDDB and Vetra CondDB. The different slices can be specify by --dddb (-d), --cond (-c) and --vetra (-v). The number of DB you give to the script **must** be equal to the number of tags, and if you give more than one slice, the numbers have to be equal.

You can give file names (two will be produced), with --file (-f) option. The first file will contain one TTree *per* DB, the second one one TTree containing compared values *per* pair of DB.

The script uses Utils.py module, included in the package to create a C++ struct and TBranch definitions. An option file is written (from options/DefaultDB.opts, included as well) and a second python script is called, which will read on DB and fill a TTree. The second script is called once per set of databases. Once all the DBs are read, CompareDBs will produce one TTree per possible comparison (if 2 DBs => 1, 3 => 2, ...).

If you want some other values to be read and compared, please modify options/Config.xml. You can add branches. The name will be the name of the leaf and of the structure field. The type is the data type (int or float), and you can decide whether the value will be compared (for instance you want to compare noise) or not (you don't want to compare TELL1 source ID, but you want to keep this information). The data field is the method used to get the value. The method in which this will be included has the following arguments :

- stTool, the STReadoutTool
- vetraTool, the VetraCfg tool
- mSector, the presently processed DeSTSector
- tell1ID, the TELL1 source ID (unsigned int)
- strip, the strip number **in the sector** (from 1 to nStrip())
- channel, the TELL1 strip number (from 0 to 3071)
- svcBox, a flat number constructed from the service box name
- stChanID, the STChannelID of the strip, which allows you to access the station, detector region, etc..
- vals, the list were all the fields you need will be stored before they are written the structure of type Data

For instance, if you want to compare the changes applied to the disabled beetles, add the line

```
<branch name="DisBeetle" type="int" comparison="yes"> vCond.disableLinksPerBeetle().at( channel /
```

Note that you don't need to retrieve the STVetraCondition yourself : a STVetraCondition named vCond is provided automatically.

Perl scripts

perl script

Allows you to put a file produced with STNoiseMonitor in its right place, taking care of the non-present sectors. The syntax is

```
DumpNoiseIntoConditions input_file output_file
```

perl script

Allows to compare two versions of the same file (xml or txt). The file is not requested to contain only one parameter (ie you can process TELL1Cond.xml for instance) you have ot specify the files names of course, the xml tag you want to compare and a tolerance. You can also specify a logfile name (the default one is "log"). The syntax is :

```
GetDBChanges.pl oldfile newfile tagname [epsilon] [logfile]
```

It's important to know which file is the *old* and the *new* one, since the logfile will tell you what was removed from the old and what was added to the new one. The tagname can be whatever you want ("Pedestal_mask", "Hit_threshold", ...). The espilon is used to do comparison :

the default value is 0. The logfile will contain all the information. For help use the "-h" or "--help" option.

perl script

Allow to customise the database comparison. It reads the Config.xml file (in options directory) and produce as an output a modified version of Utils.py.

Job options

Shared job options

The following options are common to all algorithms :

Option Name	Default value
DetType	"IT"
InputLocation	"Raw/IT/LCMSADCs"
OuputFileName	"Thresholds.txt"
WaintingTill	1000
ProduceFile	false
UnwantedTELL1s	{}
ConditionLocation	"CondDB/TELL1Board"
VetraCfgTool . DetType	TT

- DetType can be set to "IT" ot "TT", a wrong value will stop the algorithm's execution.
- InputLocation is the name of the TELL1 container.
- WaitingTill acts as ConvergenceLimit in the Emulator, use the same value as you put in the Emulator.
- ProduceFile toggle on/off the creation of the output text file.

- If the file you're using contains some TELL1 you don't care about, just add the TELL1 IDs in UnwantedTELL1s in the following way :

```
ST.UnwantedTELL1s = { 8, 49};
```

- The ConditionLocation is the path to the condition DB, which shouldn't need to be changed.
- The VetraCfgTool . DetType is important when you want to read the Vetra Condition DB. Be sure to put the right detector type, since your job will crash only at the end of the run... The concerned algorithms are : First and Second order correction, Threshold finder.

All the options related to location are now STConfigProperties, ie you don't have to set them if the only change is running over TT instead pf IT. The base class will do it automatically for you.

Header correction options

Both header correction algorithm have the same options :

Option Name	Default value
ZeroThreshold	100
OneThreshold	150
CorrectionEnabled	-
Offset	0

The first two are the threshold under (resp. above) which the header is considered to be negative (resp. positive). The ConditionLocation is the path to the conditions, which should look like 'ITCondDB/TELL1Board'. CorrectionEnabled is a *bool* value, which toggles on/off the correction. The offset is used to look at other strip (for instance the influence of the header value(s) on strip 1). The offset takes values between 0 and 31 (automatic conversion if wrong setting).

NB : the correction (ie reading values from CondDB) for STSecondCorrection is not implemented yet, and should crash at execution.

Threshold finder options

Option Name	Default value
ThresholdsAndValues	

Here you must specify which thresholds you want to set (ie Hit_threshold, Confirmation_threshold, ...) with the **same** name as it is in the CondDB, and then the signal to noise ratio you want (if the hit threshold is set to 3 sigma of the noise : put 3) and the number of channels that have a different value. All these fields have to be entered as std::string. Here is an example :

```
Finder.ThresholdsAndValues = { "Hit_threshold", "3.", "3072", "Confirmation_threshold", "4.", "4
```

It will create 2 files, named as follow : OutputFileName + "Hit" (or "Con") + sigma value. This has been done to accelerate the threshold production. You can now do something like :

```
Finder.ThresholdsAndValues           =
  {
     "Hit_threshold", "2.5", "3072",
     "Hit_threshold", "3.0", "3072",
     "Hit_threshold", "3.5", "3072",
     "Hit_threshold", "4.0", "3072",
     "Hit_threshold", "2.5", "3072",
     "Confirmation_threshold", "3.0", "48",
     "Confirmation_threshold", "3.5", "48",
     "Confirmation_threshold", "4.0", "48",
     "Confirmation_threshold", "4.5", "48",
     "Confirmation_threshold", "5.0", "48",
     "Confirmation_threshold", "5.5", "48",
     "Confirmation_threshold", "6.0", "48"
  };
```

and get in one run the 12 different files!

Scanner options

For this algorithm we have only one option :

```
 ClusterLocation
```

which default value is "Emu/IT/Clusters". This is the location in TES where the clusters are stored.

Error masker options

Here is the list of specific options :

Option Name	Default value
ErrorLocation	"Raw/IT/ErrorTELL1"
RunOnNZS	true
FullDetails	true
BoardAlias	false
FullNames	false

- ErrorLocation is the path in TES to the Error from STErrorDecoding. When running on NSZ data, the algorithm will create the banks in this location.
- RunOnNZS has to be "false" when running on ZS data.
- FullDetails allows to get a better summary, ie the algorithm shows which links / ports have bad strips.
- BoardAlias is used to get the TELL1 number corresponding to the source ID.
- FullNames is used to print the sector names instead of the link and ports numbers.

Bad link masker options

This is the list of options :

Option Name	Default value
LowGainThreshold	110.
LowNoiseThreshold	1.3
HighNoiseThreshold	6.
ShortThreshold	6.
TestPulseData	false
FullDetails	true
BoardAlias	false
KnownBadStrips	true
ReadoutTool	ITReadoutTool
FullNames	false
MaskFromDB	{ 1, 2, 4, 5 , 6, 7, 9, 10 }

- LowGainThreshold is applied to tag low signal strips. If the mean value of the strip is under this value, the strip is tagged as "Low Gain".
- LowNoiseThreshold tags the strips where the signal level is OK, but the noise is too small. We know that the overall noise (sensor + beetle + ...) is around 2, that's why I guess something under 1.3 is too

low.

- HighNoiseThreshold tags noisy channels. As the raw data has no header correction, it doesn't tag the first strip of an analog port.
- ShortThreshold is used to tag a pair of shorted strips.
- TestPulseData must be set to true when running on TP data, this allows to look for shorted strips.
- FullDetails allows to get a better summary, ie the algorithm shows which links / ports have bad strips.
- BoardAlias is used to get the TELL1 number corresponding to the source ID.
- KnownBadStrips toggles on / off the reading of known bad links / ports / strips, according to the lab's results. It has to be turned off if you want to compare the found bad strips with the known ones.
- ReadoutTool is used to get known bad strips. The conversion to TT readout tool is automatic, you don't have to worry about it.
- FullNames is used to print the sector names instead of the link and ports numbers.
- MaskFromDB are the status that will be masked in the TELL1Cond.xml. The default value masks everything but 'OK' and 'Pinhole' status. Another useful value is just 4, which stands for 'ReadoutProblem'.

Pedestal writer options

There is no specific options.

Noise monitoring options

This is the list of options :

Option Name	Default value
ConstantFactor	776.
SlopeFactor	47.9
NbrDigits	2
ElectronsPerADC	false

- The constant and slope factors are used to compute the number of electrons due to noise from the capacitance C ie :
  where the constant factor is b and the slope is a.
- The number of digits is the number of decimals that will be stored in the DB.
- The ElectronsPerADC bool value toggles on / off the production of the ElectronPerADC value in the output file.

-- JohanLuisier - 31-Mar-2010

This topic: LHCb > LHCbSTVetraAnalysisOld
Topic revision: r1 - 2010-03-31 - JohanLuisier

Latex rendering error!! dvi file was not created.