TASCA Go4 Analysis

OpenOffice document tascaGo4intro.odt (H.Essel, 26. June 2009) SVN rev. 307

Setups

Set up account

The tasca account should be customized for more convenience. One should define a variable for the repository path:

export SVN=https://subversion:443/goofy/go4/applications/tasca

To create a new working copy of the repository, create a directory and

```
mkdir myws
svn checkout $SVN myws
cd myws
svn info
```

Then one can use svn commands like

```
svn list $SVN
```

to get a listing of the subversion repository. Some useful alias:

```
svndiff='svn diff --diff-cmd /usr/bin/diff -x "-EwbB" '
svndiffl='svn diff --diff-cmd /usr/bin/diff -x "-qEwbB" '
```

On a workspace directory these give a list of files different from repository (second line file list only).
Above has been added to .bashrc file (HE). Other useful alias can be defined here.

Set up working directory

Once the directory is made an svn working directory (by checking out a repository to it) there are few commands to deal with the repository:

svn info
: show the repository the workspace belongs to

svn list $SVN
: list of repository

svn update
: update workspace from repository

svn commit -m “enter here comment” [file]
: copies all changed files to repository. If a file is specified, only this file is copied (if modified).

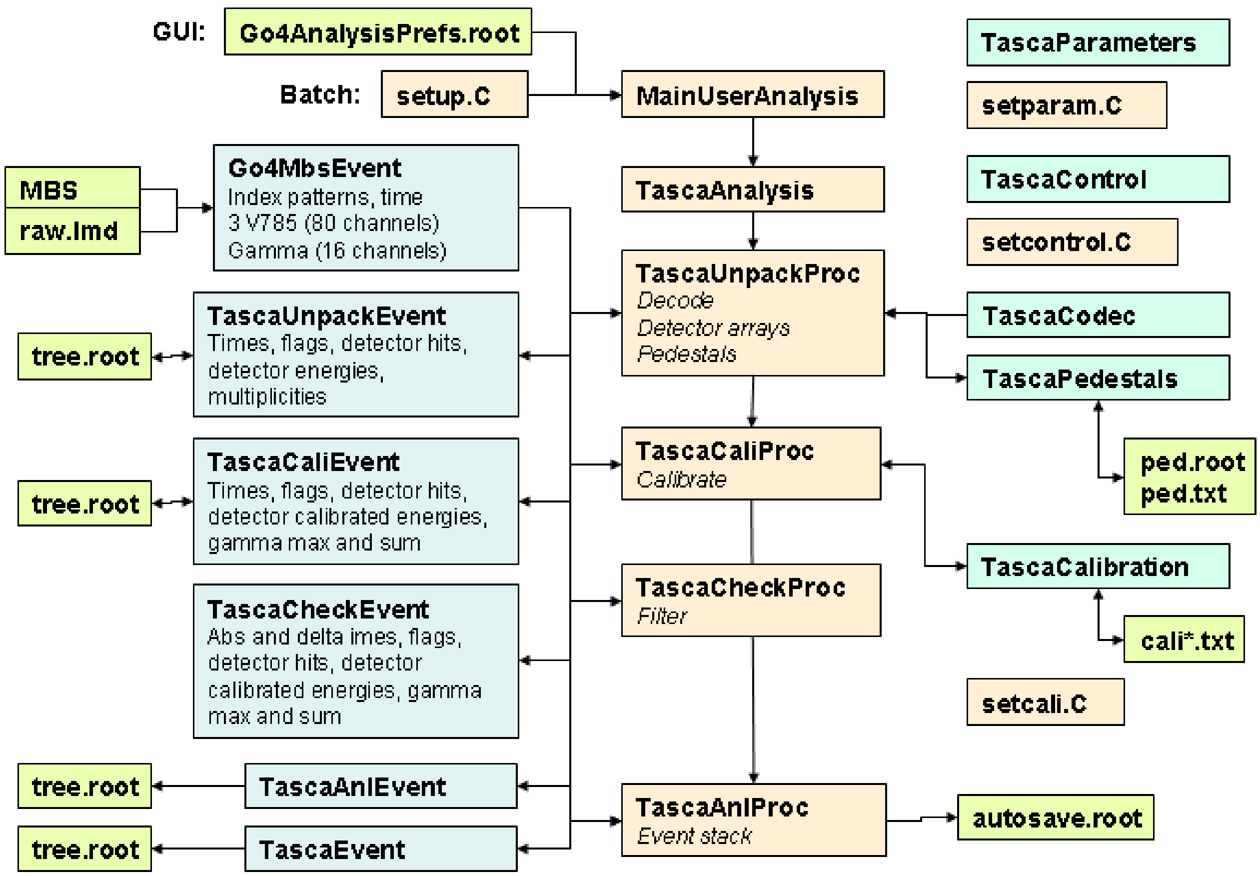
After login

Setup everything for Go4 (**now already done in .bashrc**)

. go4login 402-00

. lealogin

(Note the space behind the dot.)



Go4 analysis steps

The Go4 analysis

To build the Tasca analysis, simply:

make

The executable made is

MainUserAnalysis

It can be called from shell or is started from GUI. In principle it does the same in both cases.

Batch mode

The analysis is steered by a ROOT macro file setup.C. You can edit this file before running the analysis. There are the following lines:

```
TString unpackProcess("yes");
TString unpackStore("no");
TString unpackOverWrite("yes");

TString caliProcess("yes");
TString caliStore("no");
TString caliOverWrite("yes");

TString checkProcess("yes");
TString checkStore("no");
TString checkOverWrite("yes");

TString analysisProcess("yes");
TString analysisStore("no");
TString analysisOverWrite("yes");

TString autosave("yes");
Int_t autosaveinterval=0; // after n seconds, 0 = at termination of event loop
```

Examples:

MainUserAnalysis -f file.lmd
MainUserAnalysis -f @file.lml

processes file or list of files. respectively.

MainUserAnalysis -t r4-4 10000

connects to MBS transport node R4-4 and processes 10000 events.

Usually in batch mode one either writes an auto-save file (containing all histograms, parameters, etc.), and/or any event file. The auto-save file name and the event file names are prefixed by the input file or node name

b_r4-4_AS.root, b_r4-4_Unpacked.root, b_r4-4_Calibrated.root, b_r4-4_Checked.root, b_r4-4_Analysis.root

The b_ is added in batch mode only. Any of these can be opened by ROOT or in the GUI. To process these in batch:

MainUserAnalysis -f r4-4

The pre and postfixes are added automatically.

To process files from a data directory, the variable

export TASCASTORE=/data.local3/x/x/x

must be set. Then all files are read and stored from/to it. Currently no files can be stored on a directory different from the source directory.

Interactive mode

In interactive mode the analysis is started by the GUI. In this case, the file name prefix is the analysis name specified in the Start Client panel. This name is saved by Save Settings. In addition the prefix b_ is changed to i_. Further setup is specified in the configuration panel coming up after starting the analysis. Default settings are the ones from setup.C. This setup can be modified interactively and can be stored (NOTE: after Submit!) in

Go4AnalysisPrefs.root

from where it is retrieved next time the analysis is started. If this file is present, the settings from setup.C are overwritten.

The analysis steps

The analysis is divided into four steps as shown in the figure.

Unpacker step

Input: LMD file or MBS (transport, stream server, event server)

Output: ROOT tree with values of all detector channels and detector hit lists. Details in TascaUnpackEvent.h

Autosave: Controls, Parameters, Pedestals and Codec
Histograms in directory Unpack: Adc_nn GammaE_n GammaT_n Pedestals Contents AdcAllRaw AdcAllCal TraceRaw_nn TraceE_nn Hist_nn Pileup_nn

Processing: TascaUnpackProc constructor creates the parameters, histograms and pictures. Method *TascaUnpack* uses parameter class TascaCodec to decode Adc values, gamma values, and fills the data fields of TascaUnpackEvent. TascaCodec also contains the mapping tables for the multiplexed channels.

Calibrator step

Input: TascaUnpackEvent (from Unpack step or from file)

Output: ROOT tree with calibrated values of all detector channels and gammas. Hit indices of all detectors and their values. Details in TascaCaliEvent.h

Autosave: Controls, Parameters, Calibration, CaliFitter
Histograms in directory Cali: All detector channels, gamma channels, Sum of detector channels.

Processing: Filling histograms and TascaCaliEvent data fields.

Checker step

Input: TascaCaliEvent (from Unpack step or from file)

Output: ROOT tree with calibrated hits. Hit indices of all detectors and their values.
Condition filters: EvrH, AlphaL, Alpha1L, Alpha2L, Fission1H, Fission2H, BackH
Limits set in setparam.C
Details in TascaCheckEvent.h

Histograms in directory Check: 2d histograms of stop detector (Energy-Xstripe) for each Ystripe.
Autosave: Controls, Parameter

Processing: Filling histograms and TascaCheckEvent data fields.

Analysis step

Input: TascaCheckEvent (from Checker step or from file)

Output: ROOT tree with data from TascaAnlEvent.h (currently none) or TascaEvent.h

Autosave: Creates parameters Controls, Parameters
Processing: Looking for chains, Create plain ROOT tree from TascaEvent

Control files

There are some ROOT macro files to setup several parameter values.

setcontrol.C : Lines to change:

```
  fControl->writeChainTree =kTRUE; // used by Analyzer
  //fControl->ChainCounter =0; // used by Analyzer. Without Autosave: will be 0
  fControl->UnpackHisto  =kFALSE; // used by Unpacker
  fControl->CaliHisto    =kFALSE; // used by Calibrator
  fControl->CheckHisto   =kFALSE; // used by Checker
  fControl->AnlHisto     =kFALSE; // used by Analysis
  fControl->checkTof     =kFALSE; // used by unpacker
  fControl->checkChopper =kFALSE; // used by unpacker
  fControl->checkMacro   =kFALSE; // used by unpacker
  fControl->checkMicro   =kFALSE; // used by unpacker
  fControl->TofMustbe    =kTRUE;  // used by unpacker
  fControl->ChopperMustbe=kTRUE;  // used by unpacker
  fControl->MacroMustbe  =kFALSE; // used by unpacker
  fControl->MicroMustbe  =kFALSE; // used by unpacker
```

setparam.C : Lines to change:

```
// Used by Checker
// Energy windows MeV
       Float_t EvrHmin     = 4.000,  EvrHmax     = 15.000;
       Float_t Alpha0Lmin  = 9.800,  Alpha0Lmax  = 10.200;
       Float_t Alpha1Lmin  = 9.700,  Alpha1Lmax  = 10.100;
       Float_t Alpha2Lmin  = 8.970,  Alpha2Lmax  =  9.3700;
       Float_t Fission1Hmin=60.000,  Fission1Hmax=220.0000;
       Float_t Fission2Hmin=60.000,  Fission2Hmax=220.0000;
       Float_t BackHmin    =10.000,  BackHmax    = 80.000;
// Time windows sec
       Float_t fAlphaTmin   =0.,     fAlphaTmax   =900.;
       Float_t fAlpha1Tmin  =0.,     fAlpha1Tmax  = 20.;
       Float_t fAlpha2Tmin  =0.,     fAlpha2Tmax  =180.;
       Float_t fFission1Tmin=0.,     fFission1Tmax=900.;
       Float_t fFission2Tmin=0.,     fFission2Tmax= 70.;
...
       fp->shift=5;                // Unpacker gamma decoder for energies
       fp->Adc80TofMin=300;        // signals Tof (instead of TOF register)
       fp->AdcThreshold=100;       // Unpacker uses this is minimum raw value
       fp->EventStackSize=100000;  // used in Analysis
       fp->AlphaMaxL=16000.;       // Calibrator take low value up to this limit. Above
       fp->AlphaMaxH=30000.;       // take high value up to this limit as low
       fp->AlphaMinL=1000.;        // Unpacker raw minimum value for alpha
       fp->AlphaMinH=1000.;        // Unpacker raw minimum value for alpha
```

`setcali.C` steers the calibration:

```
  fCalibration->EnableCalibration(kTRUE);    // use calibration or not
  fCalibration->SetPrefix("cali2");          // prefix for coefficient files
```

Processing LMD files

To process several LMD files at once and store the results in one root file, one must create a text file with extension .lml and specify this file preceded by an @ instead of the LMD filename. The runbatch.sh script does that on the fly (see below). File names are t018fRRRFFFF.lmd, where RRR is the run number, FFFF the file number.

Example t018f0790.lml

```
/data.local1/tasca/t018f0790381.lmd
/data.local1/tasca/t018f0790382.lmd
/data.local1/tasca/t018f0790383.lmd
/data.local1/tasca/t018f0790384.lmd
/data.local1/tasca/t018f0790385.lmd
/data.local1/tasca/t018f0790386.lmd
/data.local1/tasca/t018f0790387.lmd
/data.local1/tasca/t018f0790388.lmd
/data.local1/tasca/t018f0790389.lmd
```

I recommend to process in batch mode Unpacker and Calibration steps from one file set into one root file. Then run Checker from this root file. Append output of all inputs (output files from one file set of 4 GB are few 10 MB). Resulting ROOT file can be fast scanned by Analysis step.

It might be necessary to find events by event number in LMD files. For this purpose in each event the run and file number is stored (Run is high two bytes, file number low two bytes). In the ROOT files these events can be found easily via macros like filter...C or print...C macros. If one wants to create an LMD subset,

Create the LML files by changing into LMD file directory, then:

lmlrmake t018f 3 146

This creates files t018fRRR.lml with RRR=003 to 146 containing lists of files t018fRRR*.lmd including full path.
Create the LMD directory files by command:

lmdirmake <directory of LMD files>
lmdirmake -f file

The second command processes only one file. Search for events by command:

lmdirshow <directory> [event number]
lmdirshow -f file [event number]

Again the second command checks only one file.

LMD files have been moved to directories

/d/ship01/tasca/t018/badfiles
/d/ship01/tasca/t018/backup
/d/ship01/tasca/t018/calibration
/d/ship01/tasca/t018/targettest

Because working directly from /d was incredible slow, we first copy the data to local disk, then process, and remove the LMD files (from local disk). The place for the processed ROOT files and LMDIR files is on lxg0708:

/data.local3/offlinedata
/u/tasca/GO4_offline_t018/data

second being a soft link to the first for convenience.

GO4 analysis is in directories of

/u/tasca/GO4_offline_t018

The code for the actual batch run is in checked01, data on data/stepdata/lmdir,calibrated0x,checked0x. There is also a shell script to execute:

runbatch.sh first last

First and last are numbers xxx mentioned above.

collectchecked.C(dirfile,rootfile,events)

root -b -l "collectchecked.C(\"p01.list\",\"b_p01_Checked.root\",0)"

copies all checked ROOT files from a container text file into one. Additional filters could be applied.

filtercheckedY.C

copies all checked ROOT files with fast filter. Similar to collectchecked.C but uses partial read. One event cane be printed by

printcheckevent.C

root -b -l "printcheckevent.C(\"b_p01_Checked.root\",event)"

Analysis chain

1. Produce ROOT files with calibrated and checked events. All LMD files of a run go into one ROOT file.
   Adjust runbatch.sh script to the correct directories. In setup.C activate the Unpacker, Calibrator, and Checker. Activate output for Calibrator and Checker.
   time runbatch.sh 196 206 >> runbatch196-206.log
2. Collect ROOT files with checked events into phase ROOT files like phase p04:
   time root -b -l "collectchecked.C(\"t018p04-196-206.list\",\"../data/stepdata/checked03/b_p04_Checked.root\",0)"
3. Run GO4 Analysis to search for chains. In setcontrol.C parameter writeChainTree steers the production of ROOT tree file with the chains named xxx_Chains.root, where xxx is the first name part of the input tree file. In setup.C Deactivate all steps and activate Analysis.
   ./MainUserAnalysis -f p04 >> chainsSFoffp04.log
4. To get a complete printout of the data of a chain, use
   printevent.C(rootfile,chain number)
   root -b -l “printevent.C(\”b_p04_Chains.root\",23)"

Calibration

An automated generation of calibration coefficient files is done by macro

makecali.C(prefix, rootfile)

root -b -l “makecali(\”test\”,\”test_AS\”)”

where prefix is a string used as prefix for all file names generated, rootfile is the name of the ROOT file containing the histograms (given without trailing .root). The macro should be adjusted. Several parameters can be set inside.

Histograms/Cali/StopXL:	prefix_StopXL[144]
Histograms/Cali/StopYL:	prefix_StopYL[96]
Histograms/Cali/StopXH:	prefix_StopXH[144]
Histograms/Cali/StopYH:	prefix_StopYH[96]
Histograms/Cali/BackH:	prefix_BackH[64]
Histograms/Cali/BackL:	prefix_BackL[64]
Histograms/Cali/VetoH:	prefix_VetoH[16]
Histograms/Cali/VetoL:	prefix_VetoL[16]
Histograms/Unpack/GammaE:	prefix_GammaE[8]
Histograms/Unpack/GammaT:	prefix_GammaT[8]

The format of the calibration files is:

name value

The format of the generated files is:

name index a0 a1 a2 : NOF ChiSquare

Class TascaCalibration is the parameter class holding the coefficients. This parameter is used in the TascaCaliProc processor of the second step.

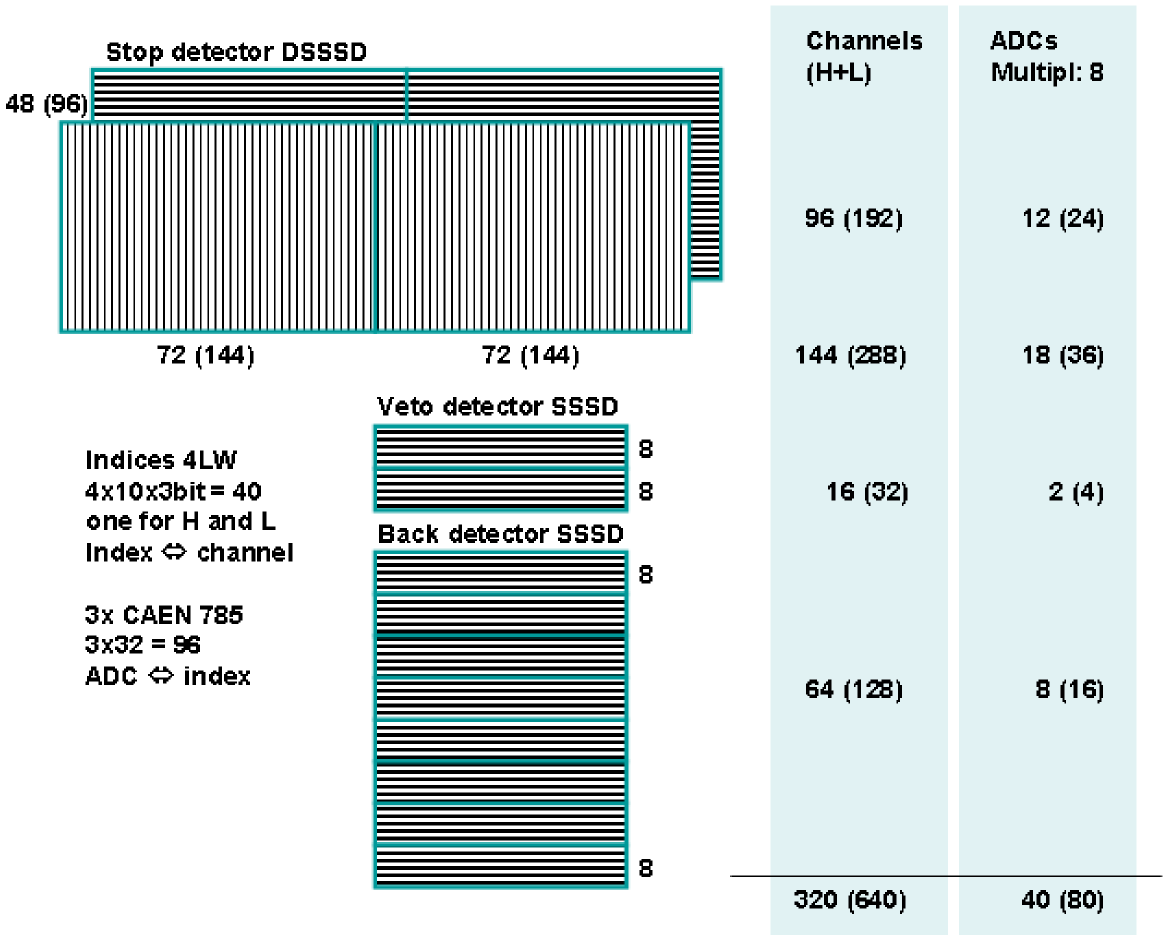
To enable/disable the calibration the macro

setcali.C

must be edited. If enabled, it reads the files produced by makecali. For these the prefix string must be set.

Class TascaCaliFitter is a parameter class with the purpose of doing the calibration interactively. This might be necessary if the automatic calculations do not work for a histogram. This parameter is used in the TascaCaliProc processor of the second step. Calculating calibration parameters is done in two steps. First we need a histogram with the measured lines and a text file with the energies of these lines. These are present in arrays inside the parameter. First fitter LineFitter is used to find out true channel numbers for corresponding lines in calibration spectrum. This fit should be done interactively on the GUI side:

– Get parameter CaliFitter from analysis (Doubleclick)
– Display calibration spectrum.
– Double click on the LineFitter fitter in the parameter editor. Fit panel will open showing the current settings of the fitter. Press Use pad of the fit panel to assign this fitter to the view panel containing the calibration spectrum and Rebuild button.
– Use peak finder 3 to find the peaks. Enlarging the noise factor removes peaks as well as minimum noise.
– Do Fit. If the positions of the lines are fitted correctly, copy the fitter back to the calibration parameter: right mouse button click on LineFitter, select Get from FitPanel .
– Check if the name of the calibration file is correct.
– Set DoFit variable to 1 (will be set back to 0 after the fit).
– Now press left arrow button. This will perform fit of the calibration curve (polynomial of order 2) in the *UpdateFrom()* method of TascaCaliFitter on the analysis side.
– Pressing right arrow button will get the results of the calibration, present in the polynomial coefficients fdA[0]...fdA[2] and in the Calibrator fitter.
– The corresponding TGraph is UserObjects/CaliGraph and is displayed by double click. Then double click on the Calibrator fitter in the parameter editor to open in a fit panel, press Use Pad, Rebuild and Draw. This will draw the calibration polynomial over the points which indicate the energy/channel of the calibration lines.



The detector layouts