D0 Note 4570

Version 0.5

TrigSimCert
A New Package to Certify and Analyse Triggers

Camille Bélanger-Champagne[a,b], Yann Coadou[a], Dugan O'Neil[a]
Steven Beale[a,c]

[a] *Simon Fraser University*
[b] *now at McGill University*
[c] *now at York University*

October 17, 2004

Abstract

A new package, trigsimcert, is presented. It can be used as a certification tool for new releases or trigger lists, or as the basis for trigger studies. It produces a ROOT tree containing information about the whole DØ trigger system. The data is saved in a structured tree, relying on classes for each object. These classes are described, as well as how to produce a tree and how to analyse it. [work in progress, not yet complete]

Contents

1 Introduction

The original purpose of the trigsimcert package is to certify new trigsim releases and to test new trigger lists. To this end, it can run on all D0om data formats currently available, as described in section 2.

It relies on the existence of two chunks containing information about all three levels of the trigger system, to produce ROOT trees containing this information in a structured way based on classes. A detailed description of the content of these classes is given in appendix A.

The ROOT trees can then be analysed using the provided macros. Their content and use, as well as advice to customise them, are described in section 3. It contains enough information to allow any user to use the macros and classes to perform his/her own trigger study. A few relevant examples of how to run the macros are reported in appendix C.

The latest documentation for TrigSimCert can always be found on the web from the main TrigSimCert page:

http://www-d0.fnal.gov/computing/trigsim/cert/trigsimcert.html

It contains links to descriptions, tutorials on how to run trigsimcert and how to use the macros, as well as results from certification of new releases and trigger lists.

2 Inputs

2.1 Input File Formats

TrigSimCert can read information from thumbnails or DST's. It uses the L1L2Chunk and the L3Chunk, which are available in both formats. For Level 3, some of the information may not be available in the thumbnail. These input files can come from reconstructed data, trigsimed data or trigsimed Monte Carlo.

There is a d0tools script, runTrigSimCert. By default TrigSimCert assumes that it is running on thumbnails. The minimal set of options to use is:

```
setup d0tools
setup D0RunII p17.00.00
runTrigSimCert -filelist=MyThumbnailFiles.dat
```

where `MyThumbnailFiles.dat` is a text file with a list of thumbnail input files.

If the input files are DST's, the `-DST` option should be passed to the d0tools script:

```
runTrigSimCert -DST -filelist=MyDSTFiles.dat
```

This option would also run fine on raw data if only the Level 3 information was required, as raw data contains an "online" L3Chunk. It would not, however, provide any information about Level 1 or Level 2 because the L1L2Chunk is not present. In order to solve this possible problem another option (`-RAW`) was introduced, that makes TrigSimCert produce the L1L2Chunk on the fly so that it is available for the analysis part to read:

```
runTrigSimCert -RAW -filelist=MyRawDataFiles.dat
```

TrigSimCert is also compatible with SAM. Using `-defname` instead of `-filelist` will automatically use the SAM manager:

```
runTrigSimCert -defname=MyThumbnailDataset
```

As all d0tools scripts an online help is available:

```
runTrigSimCert -h
```

2.2 RCP Parameters

The number of RCP files has been voluntarily limited to a minimum. Most cases are dealt with automatically but some common situations will still require the user to edit one of those files.

2.2.1 Framework RCP File

The framework RCP name convention is runD0TrigSimCert<_SAM><_format>.rcp, where _SAM is present for the SAM RCP and _format is nothing for thumbnail, _DST or _raw. The only difference between the DST and thumbnail RCP's is the unpacking of the thumbnail chunk. In the raw data mode, it also calls l1l2chunk.rcp which runs the online l1l2unpacker and l1l2_reco in "data" mode. Which framework RCP to call is decided by the d0tools script, depending on the command line options.

2.2.2 CertAnalyze RCP File

The CertAnalyze.rcp file controls what TrigSimCert is going to do. It contains three boolean flags (`certL1`, `certL2` and `certL3`), by default set to true, to decide which trigger levels should be analysed.

A second part concerns the output file. Its name can be changed (trigsimcert.root by default), as well as the compression level (default: 1) and the `autosave` option (by default turned off): if it is set to a positive integer n, the ROOT tree will be saved every n events. If the program crashes, all events that had been processed before the last autosave are still readable.

The last parameters of this RCP file specify which chunks to look at. One can separately decide which L1L2Chunk (`L1L2ChunkChoice`) and which L3Chunk (`L3ChunkChoice`) to read. By default they are both set to "default", which means "online" on data (raw or reconstructed) and offline on Monte Carlo. One can specify "online" or "offline". In order to read the "offline" chunks of trigsimed data, one has to set those parameters to "offline" by hand.

Starting with p17, it is now possible to label the L1L2Chunk and have multiple chunks in the data. All chunks generated with pre-p17 versions have a blank label, "". The "auto" mode deals with it properly, but if one specifies "online" or "offline" this mechanism is turned off, assuming the user know what he/she is doing.

The D0TrigSimCert_x.out file lists all the parameters that have been used. The D0TrigSimCert_x.log file contains information about which L1L2Chunk and L3Chunk have been attempted, depending on the RCP request, and if it succeeded or not.

2.3 L1L2Chunk

All information about Level 1 and Level 2, as well as part of the trigger bits and names information, is extracted from the L1L2Chunk. The p17 version of this chunk is much more developed than it used to be, and TrigSimCert makes use of most of its variables.

Starting with p17 this chunk can be labelled at will. By default it is "online" when produced by d0reco on data, and "offline" when produced by trigsim on data or Monte Carlo or by d0reco on Monte Carlo. In both cases it contains the same information, including the trigger bit names coming from the database in d0reco or from the level3.sim file when running trigsim.

A chunk selector is used in TrigSimCert to get the requested chunk. For pre-p17 L1L2Chunk's, there was no label and therefore no selector. The only way to access this chunk is if there is only this one present in the data or by asking for a chunk with no label. TrigSimCert deal with it automatically.

In p17 you can also label the L1L2Chunk as you want. You can retrieve it in trigsimcert as described in section 2.2.

2.4 L3Chunk

The L3Chunk is available as is in raw data and in DST's. In the thumbnails a trimmed down version is saved. When unpacking the thumbnail the L3Chunk is recreated but some information is missing. This may show up in some variables being always at 0 when running on the thumbnail while they have meaningful values when running on DST's. Hopefully all the regularly used variables should be available in both formats.

The L3Chunk has always had a hard coded label: "online" for raw data and "offline" for trigsimed data or Monte Carlo.

3 Output

The output of TrigSimCert has the form of a ROOT tree. Each branch of this tree corresponds to a trigger object, for example a L1Cal Trigger Tower or a L3Muon. This tree is absolutely independent of the release of the D0 software except for the `kinem_util` package. The classes used to fill the tree are the same ones that allow to extract information from it in ROOT macros. An example of that is in the macros provided in the `macros` directory of the `trigsimcert` package. A detailed description of the classes of TrigSimCert can be found in appendix A.

3.1 Analysis Macros

The macros of TrigSimCert produce a set of plots showing the distributions of various properties of each trigger object in the output of TrigSimCert. The macros can be used in two different ways: as macros run within ROOT or as a command line executable via a Makefile. Both have the same functionalities. The executable can be run through a real debugger like ddd, which is not the case for the ROOT macros, and one gets gcc compilation messages that are more useful than the ACLiC ones.

The macros can serve two purposes: allowing the user to save a series of histograms of the properties of trigger objects for a TrigSimCert output tree to compare two of those trees by using one as a reference set. The other set, called "data" is plotted on top of the reference set, the reference histogram is scaled to the area of the data histogram and the resulting combined histogram then allows for a visual comparison. A reduced χ^2 is computed for the comparison between the histograms. The histograms produced by the macros are available in two formats: a ROOT file and a PostScript document.

3.1.1 General Description

Both the ROOT macros and the executable require as argument a text file, the file list, which contains a list of TrigSimCert tree files. The output directory to be used can also be specified (optional for the command-line executable which takes the current directory by default). The main macro class is called `Plots`. The `Plots` constructor opens the files listed in the input file list and adds them to a ROOT TChain. If none of the files listed exist, an error message is issued and the constructor exits. If only some of them exist, it prints an error message with the name of each file it could not open and proceeds to analyse the ones it could find. It also checks if the output directory listed exists by creating a file named "test" and putting it into that directory. If it does not exist, the constructor exits. A third argument (optional) allows the user to name the output PostScript file. The output ROOT file name is constructed by replacing ".ps" by ".root". By default this name is set to "plots.ps", which makes the default ROOT file name "plots.root". The constructor also checks the filename provided by the user and completes it with the extension if necessary. Using the same name twice will result in the existing file to be overwritten without a warning being issued.

The constructor calls the `Init()` method which instantiates a TClonesArray for each branch of the TrigSimCert tree. It also creates a set of new empty histograms for the branches.

The `Plots` TChain is read and the histograms are filled through the `Loop()` method. Once a TrigSimCert object has been extracted, it is passed to the corresponding histograming object which fills the appropriate histograms. At trigger levels 1 and 2 this means filling a predetermined number of histograms for each object. At level 3, a set of histograms is defined for each trigger object (for example, L3 jets can have : number of jets, E_T, η and ϕ histograms) but many tools can be called for the same trigger object, depending on the trigger list used. Therefore, one set of trigger object histograms will be filled for each tool instance. That difference is reflected in the histogram filling classes of level 3 objects which are more complex than the level 1 and level 2 ones, to be able to handle the variable number of histograms to be produced. Namely, the histograms are stored in vectors and new sets of histograms are created as new tools are found in the course of the `Loop()` function call.

Most of the trigger objects (muons, electrons, jets, etc.) can be found more than once in each event and internal loops ensure they are all used when filling the histograms.

After the `Loop()` function has run, one of two things can happen: a reference file is created or the comparison plots are produced. If the ROOT macro was run using `Loop(true)` or the executable was run using the -makeref command line option, a reference file is created (it is always named "Ref.root") and the `Save(*file)` function is called on each histogram class.

If only `Loop()` is run (the argument is set to false by default) or if the -makeref option is not used, when `Loop()` has produced its histograms it looks for a file named "Ref.root" in the specified output directory. If it exists, a PostScript file of the given name is created and the `Draw` function is called on the various histogram classes to produce comparison plots. Histograms are also saved in a ROOT file of the same name as the PostScript file, in the same format as the Ref.root file.

Complete example sessions and list of options can be found in appendix C.

3.1.2 Histogram Classes

The histogram classes all inherit from a general histogram plotter class called `HDraw` located in the `macros/HistoFill` directory. The most important feature of this class is the `DrawHisto` function which is called in all inheriting histogram classes when its histograms are drawn. The calling line is

```
DrawHisto(TH1F *DATA, bool drawref, TH1F *REF, TLegend *Leg, char *hist_name,
    char *file_name, bool Gaussian=false, bool isLog=false, bool isRangeAdj=false)
```

DATA is the histogram containing the data set. The `drawref` variable is set to true if the reference set is to be used to draw comparison plots. REF is the histogram containing the reference set. Note that even if `drawref` is set to false, there must be a REF argument in the calling line. `Leg` is a legend for the histogram to which items can be added. It allows passing of an existing legend with entries that will not get overwritten. The strings `hist_name` and `file_name` are respectively the name of the histogram being drawn and the histogram class it is originating from. They are mostly present so they can be printed to screen in case of problems with the drawing of the histograms. `Gaussian` indicates if the histogram distribution should have a Gaussian curve fitted to it, in which case the RMS is given in the legend of the histogram (false by default). The `isLog` variable indicates whether the histogram is to be plotted with a logarithmic y-axis scale (false by default). The `isRangeAdj` argument allows the drawing method to attempt to modify the range of the x axis to improve the legibility of the histogram (false by default). If set to true, the x axis is rescaled to half of its original range if 98% of the distribution is in the left, middle or right half of the histogram.

The various specific histogram classes all contain, on top of a constructor and a destructor, a `Fill`, a `DoneEvent`, a `Draw` and a `Save` functions. For L1 and L2 classes, they are all very straightforward functions. The constructor creates the appropriate set of histograms for the histogram class and the destructor deletes them. `Fill(SomeTrigsimcertClass *tobject)` fills the histograms and, if appropriate, increments a counter that will indicate at the end of the event the number of instances of the trigger object studied in the given event. The argument `tobject` is an instance of some trigsimcert class (the appropriate header files can be found in the `trigsimcert` directory of the TrigSimCert package). The `DoneEvent()` method, if appropriate, fills the histogram containing the number of trigger objects of that class in the event and resets the counter to zero. The `Save(*file)` function saves the histogram to the file given as argument. The `Draw(TFile *ref, TPostScript *ps, bool debug=false)` method creates the TCanvases and TPads necessary to draw the given set of histograms and the corresponding TLegends. It calls the `DrawHisto` function of the base class to produce the histograms and writes the histograms to the PostScript file. In the arguments passed to the Draw method, `ref` is the file containing the reference plots, `ps` is the output PostScript file name and the `debug` parameter, when set to true, keeps the canvases open to facilitate the debugging of the macros.

In the case of L3 histogram classes, a set of histograms is produced for each physics tool found in the sample for this histogram class. To keep all these histograms in memory, they are stored in vectors of histogram objects. Every time a tool gets called, the `GetHisto(toolname)` function, where `toolname` is the string name of the tool called, returns the index of the set of histograms associated with that tool. If there is no such set of histograms, a new set is created and added to the vector. The rest of the class for the L3 histograms works exactly like the L1 and L2 classes.

3.1.3 Histogram Properties

Four histograms are plotted by page. The set called Reference is presented as a solid red line and Data as black crosses. The Reference is scaled to the area of Data. If there are any entries outside the range of the histogram, extra legend entries indicate the number of entries in the underflow and/or overflow bins for Data and Reference. The legend also includes a value of the reduced χ^2 (if it is not zero) computed on the non-zero bins of the histograms. If the Gaussian flag is set in the `DrawHisto` method, the RMS of the Gaussian fit is also added to the legend. If the χ^2 value is higher than a preset value (by default, 3), a red star appears beside it on the legend to signal an important mismatch between Data and Reference. If there is a set of histograms that exists for one of the trigger objects in Reference but not in Data, this set of histograms is simply not drawn in the output file. If the opposite happens, a set of histograms exists in Data but there is no Reference for it, the Data histograms are plotted in the output file and a warning message is issued.

3.1.4 Adding Histograms to a Class

Adding new histograms to an existing class is a straightforward process, slightly more involved for L3 classes which have a more complex structure than L1 and L2 classes.

An example of an L1 class can be found in Appendix B. L2 classes are extremely similar. To add an histogram to such a class called, for example, HISTO4, a new set of variables corresponding to the properties of the new histogram must be declared in the private members of the histogram class (defined in a .C file in the `macros/HistoFill` directory) as well as a new histogram object:

```
Float_t HISTO4max;
Float_t HISTO4min;
Int_t HISTO4bins;
std::string HISTO4axis;
TH1F *HISTO4;
```

In the constructor, these properties must be set and the histogram made:

```
Float_t HISTO4max=100;
Float_t HISTO4min=0;
```

```
Int_t HISTO4bins=100;
std::string HISTO4axis="the x axis name of the new histogram";
HISTO41=new TH1F("class_name_HISTO4", "title of the histogram", HISTO4bins,
                 HISTO4min, HISTO4max);
HISTO4->GetXaxis()->SetTitle(HISTO4axis.c_str());
HISTO4->GetXaxis()->CenterTitle();
```

The new histogram must be deleted in the destructor:

```
delete HISTO4;
```

It must also be filled in the histogram `Fill` method:

```
HISTO4->Fill(appropriate_accessor);
```

where `appropriate_accessor` returns the value to be put in the histogram for the event and object processed.

In the `Save` method, the new histogram must be written like the other ones:

```
HISTO4->Write();
```

Finally, in the `Draw` method, on the appropriate drawing pad a new legend must be created, the reference histogram fetched and the `DrawHisto` method called for the new histogram:

```
TLegend *Leg4=new TLegend(.75,.75,.99,.99,"");
if(drawref)
  {
    TH1F *HISTO4ref = (TH1F*) ref->Get("class_name_HISTO4");
    DrawHisto(HISTO4,drawref,HISTO4ref,Leg4,"HISTO4",THISFILE);
  }
else
  DrawHisto(HISTO4,drawref,0,Leg4,"HISTO4",THISFILE);
```

Because of their different structure, a few extra changes have to be made to the L3 classes. Amongst the private members there must be a vector of histogram objects for the new histograms:

```
vector<TH1F> HISTO4;
```

Typically, the histogram pointers are named differently in the L3 classes:

```
TH1F *HISTO4_T;
```

In the destructor, the histograms are cleared instead of deleted:

```
HISTO4.clear();
```

In the `Fill` method, there are two ways to fill the histograms and the new histograms must be added to both. If the tool has already been used (index in the vector > 0):

```
HISTO4.at(index).Fill(appropriate_accessor);
```

If a new tool is found, a new set of histograms has to be created, filled and added to the vector:

```
string l_HISTO4 = "class_name_HISTO4_"; l_HISTO4+=toolname;
HISTO4_T=new TH1F(l_HISTO4.c_str(), "Histogram title", HISTO4bins, HISTO4min,
                  HISTO4max);
HISTO4_T->GetXaxis()->SetTitle(HISTO4axis.c_str());
HISTO4_T->GetXaxis()->CenterTitle();
HISTO4_T->Fill(appropriate_accessor);
HISTO4.push_back(*HISTO4_T);
```

In the `Save` method, a for loop over the tools ensures all histograms get saved. Before the `Write()` method is called, the histogram has to be fetched in the vector:

```
*HISTO4_T  = HISTO4.at(i_tool);
HISTO4_T->Write();
```

Finally, the `Draw` method has to account for this new histogram. It contains more possible cases than for L1 and L2 histogram classes:

```
TLegend *Leg4=new TLegend(.75,.75,.99,.99," ");
if(drawref)
  {
    TH1F *HISTO4ref = (TH1F*) ref->Get(l_HISTO4.c_str());
    //Make sure this tool exists in the reference
    if(HISTO4ref!=0)
      { DrawHisto(&(HISTO4.at(i_tool)), drawref, HISTO4ref, Leg4, "HISTO4",
                  THISFILE, false, false, true);}
    else
      {
        DrawHisto(&(HISTO4.at(i_tool)), false, 0, Leg4, "HISTO4", THISFILE,
                  false, false, true);
        cout<<"WARNING: in "<<l_HISTO4
            <<" : tool not present in reference"<<endl;
      }
  }
  else
    DrawHisto(&(HISTO4.at(i_tool)), drawref, 0, Leg4, "HISTO4", THISFILE,
              false, false, true);
```

3.1.5 Creating a New Histogram Class

The simplest way to create a new histogram class is to entirely copy the structure of an existing one and adapt it to produce the correct output. The L1 and L2 classes have a structure which is different from the one of L3 classes and this has to be taken into account when choosing which class to copy and modify. The example shown here is for a L1 class.

For a new L1 class named "Hl1new" make sure to include the header file of the TrigSimCert class you are interested in by changing the first include statement. If this class is, for example, L1TSCclass:

```
#ifndef L1TSCCLASS_H_
#include "trigsimcert/L1TSCclass.hpp"
#endif
```

The names of the constructor and destructor should be changed in accordance to the new histogram class name.

Then all occurrences of the old histogram class name should be replaced by Hl1new and the histogram properties set to correct values. The declaration of the fill method must be switched over to the correct TrigSimCert class:

```
void Fill(L1TSCclass *l1tscclass);
```

and all `Fill` instances have to be switched over to the appropriate accessors for the corresponding TrigSimCert class. The variable `THISFILE` should also be switched to the correct name:

```
THISFILE = "Hl1new.C";
```

In the `Draw` method, the canvas name should also be changed to be coherent with the new class name. To modify the histograms and their content, refer to section 3.1.4.

The new histogram class must be added to the `Plots` method. In Plots.h:

```
#include "HistoFill/Hl1new.C"
```

A ROOT tree branch class must be declared and set, a TClonesArray declared as well as a histogram class object. All the lines shown here should be added where the corresponding statements are made for the other histogram classes:

```
L1TSCclass     *l1tscclass;
TClonesArray   *fl1new;
Hl1new         *hl1new;
```

Modifications should also be done to the Plots.C file in a similar manner, starting with where the class gets filled. This is generally done by:

```
for(int j=0;j<(fl1new->GetLast()+1);++j) {
  l1new = dynamic_cast (fl1new->At(j));
  hl1new->Fill(l1new);
}
```

Then continue modifying the Loop function:

```
hl1new->DoneEvent();
hl1new->Save(OUT);
hl1new->Draw(REF,ps);
```

The `Init()` function needs to create and set a certain number of objects:

```
l1tscclass = new L1TSCclass();
// the second argument is the number of objects to start with
fl1new = new TClonesArray("L1TSC",1);
fChain->SetBranchAddress("L1TSCclass",&fl1new);
hl1new = new Hl1new();
```

In the `ReInit` method:

```
delete fl1new;
delete hl1new;
```

In loadstuff.C the name of the TrigSimCert class needs to be added if it is a new class:

```
#include "trigsimcert/src/L1TSCclass.cpp"
```

and in loadstuff_linkdef.h as well:

```
#pragma link C++ class L1TSCclass+;
```

Finally the class should be added to the Makefile by putting L1TSCclass.hpp in the HDRS variable.
The new histogram class is now ready to use.

3.2 Examples of Output

When the macros are run on two sets of trigsimcert ROOT trees, a PostScript file containing comparison plots is produced. An example of such a plot is shown in Figure 1. The red histogram corresponds to the reference set and the black crosses to the new data being compared to the reference. They have their respective axes in red on the right hand side and in black on the left hand side. A χ^2 value is displayed, accompanied with a red star if the matching is not good. If there is an overflow, it is mentioned in the legend.

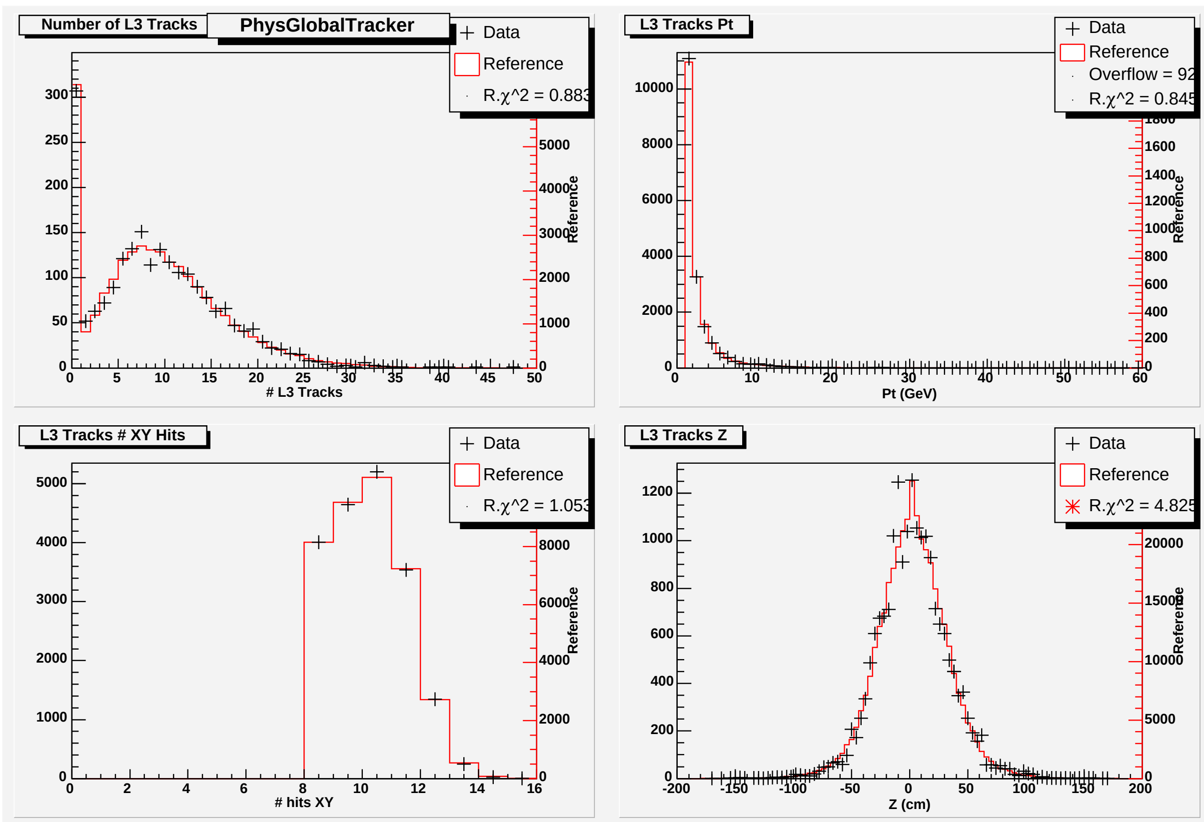

Figure 1: Example of comparison plots produced by the trigsimcert macros

A Classes description

The following tables present the classes and methods called by TrigSimCert to produce the ROOTtuple output. Unless specified, any given table corresponds to a branch of that name in the output of TrigSimCert. Each one of these branches consists of a TClonesArray of a specific TrigSimCert class. Before each table, the corresponding TrigSimCert header file with the class definition is given, as well as the analysis package that is used to fill the information to the L1L2Chunk or the L3Chunk that is read back by TrigSimCert. The exact file or web page where the information was found to document these tables is also given to facilitate more in dept research. The columns of the table give the accessors associated with each branch in the TrigSimCert output, the methods used from the respective packages to fill the branch and the physical interpretation of the various leaves contained in the branch.

A.1 Triggers

The Triggers branch is a TClonesArray of Trigger objects. It contains one entry for each L3 trigger and provide all the relevant information about the three trigger levels associated with that L3 trigger.

Header file: trigsimcert/Trigger.hpp
Packages used to fill the chunk read: l1l2_evt and l3fchunk
Information taken from l1l2_evt/L1L2Chunk.hpp and l3fchunk/L3Chunk.hpp

Accessor		
L1Name()	string	name of the L1 trigger
L2Name()	string	name of the L2 trigger
L3Name()	string	name of the L3 trigger
L1Bit()	int	L1 bit number
L2Bit()	int	L2 bit number
L1Passed()	bool	L1 trigger was fired
L2Passed()	bool	L2 trigger was fired
L3Passed()	bool	L3 trigger was fired
L1Prescale()	int	prescale of the L1 trigger
L2Unbiased()	bool	L2 trigger was unbiased
L3Unbiased()	bool	L3 trigger was unbiased
L3ForceUnbiased()	bool	not L3Passed() and either L2Unbiased() or L3Unbiased(). For DST format: status is "force_unbiased"

A.2 Event

The Event branche simply contains the run and event number.

Header file: trigsimcert/Event.hpp
Package used to fill the chunk read: edm
Information taken from edm/Event.hpp

Accessor	Method Used	

RunNumber()	collisionID().runNumber()	int	run number associated to the event read
EventNumber()	collisionID().eventNumber()	int	event number of the event read

A.3 Level 1 Triggers

A.3.1 L1Cal

The L1Cal branch contains the global information about the calorimeter as a whole.

Header file: trigsimcert/L1Cal.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l1cal_reco.hpp

Accessor	*Method used*		
GblEMEt()	sum_em_energy()	float	total energy in the EM part of the L1 calorimeter towers
GblEt()	sum_tot_energy()	float	total energy in the L1 calorimeter towers
GblMEt()	missing_pt()	float	missing transverse energy measured from L1 calorimeter towers
EM1Sort()	em_eta(int i)	float	largest EM energy value mesured in a single calorimeter tower
EM2Sort()	em_energy(int i)	float	second largest EM energy value mesured in a single calorimeter tower
EM3Sort()	em_energy(int i)	float	third largest EM energy value measured in a single calorimeter tower
EM4Sort()	em_energy(int i)	float	fourth largest EM energy value measured in a single calorimeter tower
Tot1Sort()	tot_energy(int i)	float	largest total energy value measured in a single calorimeter tower
Tot2Sort()	tot_energy(int i)	float	second largest total energy value measured in a single calorimeter tower
Tot3Sort()	tot_energy(int i)	float	third largest total energy value measured in a single calorimeter tower
Tot4Sort()	tot_energy(int i)	float	fourth largest total energy value measured in a single calorimeter tower

A.3.2 L1CalEMTwrs/L1CalTwrs

L1CalTwrs and L1CalEMTwrs are two distinct branches in the output ROOT tree. They are TClonesArray's of the same L1CalTwr class (which contains information about a single tower) so they contain the same variables which are filled by very similar methods. They return calorimeter information on a tower-by-tower basis and the towers are sorted by transverse energy in descending order. Eta() and Phi() have been remapped (see l1l2_evt/src/l1cal_reco.cpp for explanations)

Header file: trigsimcert/L1CalTwr.hpp
Package used to fill the chunk read: l1l2_evt

Information taken from l1l2_evt/l1cal_reco.hpp

Accessor	*Method used*		
Et()	em_energy(int i) tot_energy(int i)	float	total energy in a calorimeter (EM) tower
Eta()	em_eta(int i) tot_eta(int i)	float	pseudorapidity of a calorimeter (EM) tower
Phi()	em_phi(int i) tot_phi(int i)	float	azimuthal angle of a calorimeter (EM) tower
iEta()	em_ieta(int i) tot_ieta(int i)	int	pseudorapidity bin number of a calorimeter (EM) tower
iPhi()	em_iphi(int i) tot_iphi(int i)	int	azimuthal angle bin number of a calorimeter (EM) tower

A.3.3 L1CalTiles

The L1CalTiles branch is a TClonesArray of calorimeter large tiles. They are sorted by transverse energy in descending order.

Header file: trigsimcert/L1CalTile.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l1cal_reco.hpp

Accessor	*Method used*		
Et()	tile_energy(int i)	float	total energy
Eta()	tile_eta(int i)	float	pseudorapidity
Phi()	tile_phi(int i)	float	azimuthal angle

A.3.4 L1CTT

The L1CTT branch contains a summary of the information for all sectors of the CTT.

Header file: trigsimcert/L1CTT.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l1ctt_reco.hpp

Accessor	*Method used*		
NTracks()	size()	int	number of track candidates found by the CTT
CPS()		int[80]	array of number of CPS clusters for all CTT sectors, filled by CPS(int)
CPS(int i)	numCPSclus(int i)	int	number of CPS clusters (varying between 0 and 3) for CTT sector i ($0 \leq i < 80$)
Occup()		int[80]	array of occupancy for all CTT sectors, filled by Occup(int)
Occup(int i)	occupancy(int i)	int	occupancy (varying between 0 and 240) for CTT sector i ($0 \leq i < 80$)

A.3.5 L1Tracks

The L1Tracks branch consists of a TClonesArray of CTT tracks.

Header file: trigsimcert/L1Track.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l1ctt_reco.hpp

Accessor	*Method used*		
Sector()	l1trk_sector(int i)	int	sector number of the CTT track
PtBin()	pt_bin(int i)	int	p_T of the CTT track (0: p_T=1.5-3.0 GeV, 1: p_T=3.0-5.0 GeV, 2: p_T=5.0-10.0 GeV, 3: p_T >10 GeV)
LowerPt()		float	lower edge of the p_T bin for a track found by the L1CTT trigger, derived from PtBin() (value of 1.5, 3.0, 5.0 or 10.0)
UpperPt()		float	upper edge of the p_T bin for a track found by the L1CTT trigger, derived from PtBin() (value of 3.0, 5.0, 10.0 or 999.0)
Charge()	charge(int i)	int	charge of the CTT track
CPSMatch()	l1trk_cps(int i)	int	CPS track match flag (0: no match, 1: loose match, 2: tight match) for the CTT track
Occup()	l1trk_occup(int i)	int	occupancy (varying between 0 and 240) for the CTT sector in which the track was found
Iso()	l1trk_iso(int i)	int	isolation (0: not isolated, 1: isolated) of the CTT track (isolated if it is the only track in the sector and the 2 adjacent ones)
NCPS()	l1trk_numcps(int i)	int	number of CPS clusters (varying between 0 and 3) for the CTT sector in which the track was found
Phi()		float	azimuthal angle of the track, derived from Sector()

A.3.6 L1Muons

The L1Muon branch consists of a TClonesArray of L1Muon objects containing information about the muon candidates found.

Header file: trigsimcert/L1Muon.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l1muo_reco.hpp

Accessor	*Method used*		
Err()	l1mu_err()	int	muon trigger error word
CentTrigOct()	c_trig_oct()	int	number of central octants that have L1 muon candidates
NorthTrigOct()	n_trig_oct()	int	number of north endcap octants that have L1 muon candidates
SouthTrigOct()	s_trig_oct()	int	number of south endcap octants that have L1 muon candidates

CenID0()	l1mu_cen_id0()	int	central region MTM input data word 0
CenID1()	l1mu_cen_id1()	int	central region MTM input data word 1
CenID2()	l1mu_cen_id2()	int	central region MTM input data word 2
CenID3()	l1mu_cen_id3()	int	central region MTM input data word 3
CenID4()	l1mu_cen_id4()	int	central region MTM input data word 4
CenID5()	l1mu_cen_id5()	int	central region MTM input data word 5
NorthID0()	l1mu_n_id0()	int	north endcap MTM input data word 0
NorthID1()	l1mu_n_id1()	int	north endcap MTM input data word 1
NorthID2()	l1mu_n_id2()	int	north endcap MTM input data word 2
NorthID3()	l1mu_n_id3()	int	north endcap MTM input data word 3
NorthID4()	l1mu_n_id4()	int	north endcap MTM input data word 4
NorthID5()	l1mu_n_id5()	int	north endcap MTM input data word 5
SouthID0()	l1mu_s_id0()	int	south endcap MTM input data word 0
SouthID1()	l1mu_s_id1()	int	south endcap MTM input data word 1
SouthID2()	l1mu_s_id2()	int	south endcap MTM input data word 2
SouthID3()	l1mu_s_id3()	int	south endcap MTM input data word 3
SouthID4()	l1mu_s_id4()	int	south endcap MTM input data word 4
SouthID5()	l1mu_s_id5()	int	south endcap MTM input data word 5

A.4 Level 2 Trigger Preprocessors

Most L2 trigger preprocessors inherit from the trigsimcert base class L2Base. This class contains accessors Et(), iEta(), iPhi(), Eta() and Phi(). When those accessors are present in a class from the base class, they are listed at the beginning of the table and are separated from the accessors specific to that class by a double line.

A.4.1 L2EM

The L2EM preprocessor branch is a TClonesArray of EM object candidates.

Header file: trigsimcert/L2EM.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2calemp_reco.hpp

Accessor	*Method used*		
Et()	et()	float	transverse energy
iEta()	eta_bin()	int	pseudorapidity bin number
iPhi()	phi_bin()	int	azimuthal angle bin number
Eta()		float	pseudorapidity, derived from iEta()
Phi()		float	azimuthal angle, derived from iPhi()
Iso()	isofc()	float	isolation fraction
Emf()	emfrc()	float	EM fraction for the highest E_T tower of the EM object candidate

SeedEta()	ieta()	int	pseudorapidity bin number for the EM object candidate (seed tower)
SeedPhi()	iphi()	int	azimuthal angle bin number for the EM object candidate (seed tower)
SeedEtaNeighbour()	ietaN()	int	pseudorapidity bin number of the nearest neighbour tower of the EM object (2^{nd} highest tower)
SeedPhiNeighbour()	iphiN()	int	azimuthal angle bin number of the nearest neighbour tower of the EM object (2^{nd} highest tower)
SaturatedTotTower()	saturatedTotTower()	bool	the total (EM+Had) tower is saturated
SaturatedEMTower()	saturatedEMTower()	bool	the EM tower is saturated
NoNeighbour()	noNeighbour()	bool	there is no second tower above threshold in the EM object
EtaOrPhiNeighbour()	etaorphiNeighbour()	int	the eventual second tower above threshold shares the same value of the pseudorapidity (0) or azimuthal angle (1) as the highest E_T tower
PositionNeighbour()	positionNeighbour()	int	the eventual second tower above threshold has larger (1) or smaller (0) rapidity or azimuthal angle with respect to the highest E_T tower.
EtNeighbour()	etNeighbour()	float	transverse energy of the eventual second tower above threshold
EmfNeighbour()	emfrcNeighbour()	float	EM fraction of the eventual second tower above threshold

A.4.2 L2Jets

The L2Jets preprocessor branch is a TClonesArray of jet candidates.

Header file: trigsimcert/L2Jet.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2caljetp_reco.hpp and l2caljetworker/src/CalJetWorker.cpp

Accessor	*Method used*		
Et()	et()	float	transverse energy of the jet
iEta()	eta_bin()	int	pseudorapidity bin number of the jet (cluster weighted)
iPhi()	phi_bin()	int	azimuthal angle bin number of the jet (cluster weighted)
Eta()		float	pseudorapidity, derived from iEta()
Phi()		float	azimuthal angle, derived from iPhi()

CentralTwrEta()	ieta()	int	pseudorapidity bin number of the central tower of the jet
CentralTwrPhi()	iphi()	int	azimuthal angle bin number of the central tower of the jet
LeadTwrEta()	ietam()	int	pseudorapidity bin number of the leading tower of the jet
LeadTwrPhi()	iphim()	int	azimuthal angle bin number of the leading tower of the jet
SaturatedTotTower()	saturatedTotTower()	bool	one of the trigger towers in the jet was saturated
SaturatedEMTower()	saturatedEMTower()	bool	one of the EM trigger towers in the jet was saturated
ZeroETSum()	zeroETsum()	bool	the sum of the positive transverse energies is equal to zero

A.4.3 L2Muons

The L2Muons branch is a TClonesArray of muon candidates.

Header file: trigsimcert/L2Muon.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2muonp_reco.hpp

Accessor	*Method used*		
Et()		int	not relevant (not filled)
iEta()	eta_bin(int i)	int	pseudorapidity bin number of the muon
iPhi()	phi_bin(int i)	int	azimuthal angle bin number of the muon
Eta()		float	pseudorapidity, derived from iEta()
Phi()		float	azimuthal angle, derived from iPhi()
ToroidPtBin()	ToroidPt(int i)	int	p_T bin number of the muon candidate (0.25 GeV bins)
ToroidPt()		float	physical toroid p_T (ToroidPtBin()/4)
ToroidPtNegative()	ToroidPt_negative(int i)	bool	the muon candidate has negative charge
Charge()		int	-1 if ToroidPtNegative(), +1 otherwise
L1PtThres()	L1PtThres(int i)	int	L1 p_T threshold of the L2 muon candidate
L1PtSignUnknown()	L1Pt_sign_unknown(int i)	bool	L1 could not measure the charge of the L2 muon candidate
L1PtNegative()	L1Pt_negative(int i)	bool	the L1 charge is negative
L1Charge()		int	-1 if L1PtNegative(), +1 otherwise
QMask()	Q_Mask(int i)	int	quality of the muon candidate
ScTimeA()	ScTimeA(int i)	int	timing information measured on the A-layer scintillator
ScTimeB()	ScTimeB(int i)	int	timing information measured on the B-layer scintillator

ScTimeC()	ScTimeC(int i)	int	timing information measured on the C-layer scintillator
IsCentral()		bool	the muon candidate is in the central region
IsForward()		bool	the muon candidate is in the forward region

A.4.4 L2TracksSTTPT/STTIP/CTT

L2TracksSTTPT, L2TracksSTTIP and L2TracksCTT are three distinct branches in the output ROOT tree. They are TClonesArray's of the same L2Track class (which contains information about a single track) so they contain the same leaves which are filled by very similar methods. The L2TracksSTTPT are sorted by p_T, the L2TracksSTTIP by impact parameter and the L2TracksCTT by p_T from the CTT information only.

Header file: trigsimcert/L2Track.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2trkp_reco.hpp

Accessor	*Method used*		
CTTiPhi0()	cttphi0Bin()	int	azimuthal angle bin number at the DCA point using only the CTT information
CTTiPhiEM3()	cttphiEM3Bin()	int	azimuthal angle bin number at the EM3 layer of the calorimeter using only the CTT information
CTTPt()	cttpt()	float	p_T of a track candidate using only the CTT information
CTTPhi()		float	azimuthal angle, derived from CTTiPhi0()
CTTPhiEM3()		float	azimuthal angle, derived from CTTiPhiEM3()
CTTSign()	cttsign()	int	charge of the track using only the CTT information()
CTTL2Iso()	cttl2iso()	int	L2 CTT track isolation
CTTIso()	cttiso()	int	L1 CTT track isolation (0: not isolated, 1: isolated)
CTTPreshower()	cttpreshower()	int	L1 CTT track preshower match information (0: no match, 1: loose CPS match, 2: tight CPS match)
STTiPhi()	coarse_phi_bin()	int	azimuthal angle bin number at the DCA point for the track obtained from the final track fit (including STT information)
STTPhi()		float	azimuthal angle, derived from STTiPhi() and FinePhi()
STTPt()	sttpt()	float	p_T from the final track fit (including STT information)
STTSign()	sign()	int	charge of the track from the final track fit (including STT information)
DEdx()	dedx()	int	dE/dx for the track candidate
Barrel()	barrel()	int	SMT barrel number for the track candidate

Chi2()	chi2()	float	χ^2 of the track fit
FitStatus()	fitstatus()	int	STT status word (0: fit not performed, 1: fit failed to converge, 2: fit successful)
TruncLayers()	trunclayers()	int	more than 58 SMT clusters were associated with the track fitting road. Only the first 58 were considered in the fit
Topology()	topology()	int	the track candidate has 3 layers (1) or 4 layers (0)
SkippedLayer()	skippedlayer()	int	the SMT layer skipped in the fit (valid only if Topology() = 1)
IPSig()	ipsignificance()	int	significance of the impact parameter calculated by the STT fit
ImpParam()	impactparameter()	int	impact parameter calculated by the STT fit
FinePhi()	fine_phi_bin()	int	finer granularity azimuthal angle at the DCA point: each $\pi/80$ sector is divided into 16 additional bins

A.4.5 L2MEt

*****Branch not yet properly implemented in the L1L2Chunk*****
The L2MEt branch is TClonesArray of missing E_T objects.

Header file: trigsimcert/L2MEt.hpp
Package used to fill the chunk read l1l2_evt
Infomation taken from l1l2_evt/l2calmetp_reco.hpp

Accessor	*Method used*		
MEx()	ETx()	float	x component of the missing transverse energy vector
MEy()	ETy()	float	y component of the missing transverse energy vector
ScalarEt()	scalarET()	float	scalar transverse energy
PosETRing(int i)	posETring(int i)	float	positive contribution to the transverse energy in a given eta ring ($0 \leq$ i ≤ 4)
NegETRing(int i)	negETring(int i)	float	negative contribution to the transverse energy in a given eta ring ($0 \leq$ i ≤ 4)
MEtPhi()		float	azimuthal angle, derived from MEx() and MEy()
MEt()		float	missing transverse energy ($\sqrt{\text{MEx}()^2 + \text{MEy}()^2}$)

A.4.6 L2CPS

The L2CPS branch is a TClonesArray of trigger objects in the Central Preshower D etector.

Header file: trigsimcert/L2CPS.hpp
Pacakge: l1l2_evt
Information taken from l1l2_evt/l2cpsp_reco.hpp

Accessor	*Method used*		
Et()	pT()	float	transverse energy

iEta()	etaBin()	int	pseudorapidity bin number
iPhi()	phiBin()	int	azimuthal angle bin number
Eta()		float	pseudorapidity, derived from iEta()
Phi()		float	azimuthal angle, derived from iPhi()
LooseMatch()	loosematch()	int	loose track match
TightMatch()	tightmatch()	int	tight track match
MultTrackTag()	multTrackTag()	int	multiple track matches for the CPS object
IsoTrk()	isotrk()	int	isolated track
AdjacentCTTMatch()	psadj()	int	track match with an adjacent CTT sector
Sign()	sign()	int	charge (0: positive, 1: negative) for the track matching the CPS object
Charge()		int	-1 if Sign() was 1, +1 if it was 0

A.4.7 L2FPS

*****Branch not yet properly implemented in the L1L2Chunk*****
The L2FPS branch is a TClonesArray of trigger objects in the Forward Preshower Detector.

Header file: trigsimcert/L2FPS.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2fpsp_reco.hpp

Accessor	*Method used*		
Et()		float	not relevant (not filled)
iEta()	absetaBin()	int	pseudorapidity bin number
iPhi()	phiBin()	int	azimuthal angle bin number
Eta()		float	pseudorapidity, derived from iEta()
Phi()		float	azimuthal angle, derived from iPhi()
uMIP()	uMIP()	int	energy deposited in the U layer
vMIP()	vMIP()	int	energy deposited in the V layer
EtaSign()	etaSign()	int	the FPS object is in the north (0) or south (1) endcap
HiLoThresh()	HiLoThresh()	int	threshold used
IsNorth()		bool	the FPS object is in the north endcap (derived from EtaSign())
IsSouth()		bool	the FPS object is in the south endcap (derived from EtaSign()

A.5 L2 Global Triggers

Most L2 global triggers inherit from the trigsimcert base class L2GblBase, which itself inherits from L2Base the accessors Et(), iEta(), iPhi(), Eta() and Phi(). It adds to them the accessors ObjectID() and BaseObjects(). When those accessors are present in a class from the base class, they are listed at the beginning of the table and are separated from the accessors specific to that class by a double line.

A.5.1 L2GblEM

The L2GblEM branch is a TClonesArray of global EM candidates.

Header file: trigsimcert/L2GblEM.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2gblEMObj_reco.hpp

Accessor	*Method used*		
Et()	pt()	float	transverse energy
iEta()	ieta()	int	pseudorapidity bin number
iPhi()	iphi()	int	azimuthal angle bin number
Eta()		float	pseudorapidity, derived from iEta()
Phi()		float	azimuthal angle, derived from iPhi()
ObjectID()	objectID()	int	L2 global trigger identifier
BaseObjects(int i)	baseObjects (int j)	int[]	identifiers of the objects used to build the L2 global object
Iso()	isolation()	float	isolation
Emf()	emFraction()	float	EM fraction
NoNeighbour()	noNeighbour()	bool	single tower EM object
NeighbourLowET()	neighbourLowET()	bool	the EM object has a single tower because the energy of the nearest neighbour is below threshold
SeedHighEmf()	seedHighEMFrac()	bool	leading tower of the EM object is above the EM fraction cut

A.5.2 L2GblMuons

The L2GblMuons branch is a TClonesArray of global muon candidates.

Header file: trigsimcert/L2GblMuon.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2gblMuon_reco.hpp

Accessor	*Method used*		
Et()	pt()	float	transverse energy
iEta()	ieta()	int	pseudorapidity bin number
iPhi()	iphi()	int	azimuthal angle bin number
Eta()		float	pseudorapidity, derived from iEta()
Phi()		float	azimuthal angle, derived from iPhi()
ObjectID()	objectID()	int	L2 global identifier
BaseObjects(int i)	baseObjects (int j)	int[]	identifiers of the objects used to build the L2 global object
Quality()	quality()	int	muon quality (1: loose, 2: medium, 3: tight)
Prompt()	prompt()	int	time of flight (1: loose, 2: medium, 3: tight)
Sign()	sign()	int	charge of the muon (0=unknown)

ToroidPt()	ToroidPt()	float	transverse momentum

A.5.3 L2GblMEt

*****Branch not yet properly implemented in the L1L2Chunk*****
Header file: trigsimcert/L2GblMEt.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2gblMET_reco.hpp

A.5.4 L2GblTaus

*****Branch not yet properly implemented in the L1L2Chunk*****
Header file: trigsimcert/L2GblTau.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2gblTau_reco.hpp

A.5.5 L2GblMJt

*****Branch not yet properly implemented in the L1L2Chunk*****
Header file: trigsimcert/L2GblMJt.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2gblMJT_reco.hpp

A.5.6 L2GblInvMass

The L2GblInvMass is a TClonesArray of invariant mass objects.

Header file: trigsimcert/L2GblInvMass.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2gblInvMass_reco.hpp

Accessor	*Method used*		
ObjectID()	objectID()	int	L2 global identifier
BaseObjects(int i)	baseObjects (int j)	int[]	identifiers of the objects used to build the L2 global object
Mass()	mass()	float	invariant mass

A.5.7 L2GblHt

*****Branch not yet properly implemented in the L1L2Chunk*****
Header file: trigsimcert/L2GblHt.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2gblHt_reco.hpp

A.5.8 L2GblTransMass

*****Branch not yet properly implemented in the L1L2Chunk*****
Header file: trigsimcert/L2GblTransMass.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2gblTranMass_reco.hpp

A.5.9 L2GblTracks

The L2GblTracks branch is a TClonesArray of track objects.

Header file: trigsimcert/L2GblTrack.hpp
Package used to fill the chunk read: l1l2_evt
Information taken from l1l2_evt/l2gblTrk_reco.hpp

Accessor	*Method used*		
Et()	pt()	float	transverse momentum
iEta()	ieta()	int	pseudorapidity bin number
iPhi()	iphi()	int	azimuthal angle bin number
Eta()		float	pseudorapidity, derived from iEta()
Phi()		float	azimuthal angle, derived from iPhi()
ObjectID()	objectID()	int	L2 global identifier
BaseObjects(int i)	baseObjects (int j)	int[]	identifiers of the objects used to build the L2 global object
Quality()	quality()	int	track quality (1: loose, 2: medium, 3: tight)
STTFit()	sttFit()	int	STT track fit status (0: no fit, 1: fit failed, 2: fit succeeded)
IPSig()	ipSig()	int	impact parameter significance
ImpParam()	impParam()	int	signed impact parameter

A.6 Level 3 Triggers

Most L3 triggers inherit from the base class L3Base. This class contains accessors Et(), Eta(), Phi() and ToolName(). When those accessors are present in a class from the base class, they are listed at the beginning of the table and are separated from the accessors specific to that class by a double line. Although most classes inherit this base class, the quantities the base class contains are not always filled from the same methods. The ToolName() accessor allows for later separation of trigger objects by trigger list tool instances. Phi() is converted if necessary to always be in $[0, 2\pi[$.

A.6.1 L3Electrons

The L3Electrons branch is a TClonesArray of L3 electron candidates.

Header file: trigsimcert/L3Ele.hpp
Package used to fill the chunk read: l3femtools_ele_results
Information taken from l3femtools_ele_results/L3ElePhysicsResults.hpp, l3femtools/src/L3TEle.cpp and

http://www-d0.fnal.gov/d0dist/dist/releases/development/l3femtools/doc/L3TEle_overview.html

Accessor	*Method used*		
Et()	get_ET()	float	transverse energy
Eta()	get_detectorEta()	float	p_T weighted pseudorapidity
Phi()	get_kineResults()->phi()	float	p_T weighted azimuthal angle
ToolName()	l3mapIter->first	string	trigger list name of the physics tool that produced the candidate
Iso()	get_isolation()	float	isolation
Emf()	get_emFraction()	float	EM fraction
Chi2()	get_chi2()	float	χ^2
EoverP()	get_E_over_p()	float	E_T/p_T
PsMatch()	PsMatch()	bool	match between calorimeter cluster and PS hits
CpsMatch()	CpsMatch()	bool	match between calorimeter cluster and CPS hits
TrackMatch()	TrackMatch()	bool	match between calorimeter cluster and track or CPS cluster
CalTrackMatch()	CalTrackMatch()	bool	match between calorimeter cluster and track
CpsTrackMatch()	CpsTrackMatch()	bool	match between CPS cluster and track
CalCpsDmin()	get_CalCpsDmin()	float	$\Delta\eta$ between calorimeter cluster and matched CPS cluster
CpsTrackDmin()	get_CpsTrackDmin()	float	Δz between CPS cluster and matched track
CalTrackDmin()	get_CalTrackDmin()	float	ΔR between calorimeter cluster and matched track (at EM3 layer)
CalCpsDPhi()	get_CalCpsDPhi()	float	$\Delta\phi$ between calorimeter cluster and matched CPS cluster
CpsTrackDPhi()	get_CpsTrackDPhi()	float	$\Delta\phi$ between CPS cluster and matched track
CalTrackDPhi()	get_CalTrackDPhi()	float	$\Delta\phi$ between calorimeter cluster and matched track (at EM3 layer)
Em1Width()	get_Em1Width()	float	transverse shower width in EM1 layer
Em2Width()	get_Em2Width()	float	transverse shower width in EM2 layer
Em3Width()	get_Em3Width()	float	transverse shower width in EM3 layer
Em1RescWidth()	Em1RescWidth()	float	rescaled transverse shower width in EM1 layer
Em2RescWidth()	Em2RescWidth()	float	rescaled transverse shower width in EM2 layer
Em3RescWidth()	Em3RescWidth()	float	rescaled transverse shower width in EM3 layer

A.6.2 L3Photons

The L3Photons branch is a TClonesArray of L3 photon candidates.

Header file: trigsimcert/L3photon.hpp

Package used to fill the chunk read: l3femtools_photon_results
Information taken from l3femtools_photon_results/L3PhotonPhysicsResults.hpp

Accessors	*Method used*		
Et()	get_ET()	float	transverse energy
Eta()	get_detectorEta()	float	p_T weighted pseudorapidity
Phi()	get_kineResults()->phi()	float	p_T weighted azimuthal angle
ToolName()	l3mapIter->first	string	trigger list name of the physics tool that produced the candidate
Iso()	get_isolation()	float	isolation
Emf()	get_emFraction()	float	EM fraction

A.6.3 L3Jets

The L3Jets branch is a TClonesArray of L3 jets.

Header file: trigsimcert/L3Jet.hpp
Package used to fill the chunk read: l3fjettools_results
Information taken from jet_evt package and l3fjettools_results/L3JetsPhysicsResults.hpp

Accessor	*Method used*		
Et()	get_ET()	float	transverse energy
Eta()	get_detectorEta()	float	p_T weighted pseudorapidity
Phi()	get_kineResults()->phi()	float	p_T weighted azimuthal angle
ToolName()	l3mapIter->first	string	trigger list name of the physics tool that produced the candidate
EmEtFraction()	get_emETfraction()	float	E_T fraction in layers 1-7
IcdmgEtFraction()	get_icdmgETfraction()	float	E_T fraction in ICD and massless gaps (layers 8-10)
ChEtFraction()	get_chETfraction()	float	E_T fraction in Coarse Hadronic layers (layers 15-17)
HotCellRatio()	get_hotcellratio()	float	ratio of hottest to next-hottest cell in the calorimeter

A.6.4 L3MEt/L3Ht

The L3MEt and L3Ht branches are TClonesArrays of L3 MEt and Ht objets.

Header file: L3MEt.hpp
Package used to fill the chunk reads: l3fCalMEt_phys_results / l3fJetMEt_phys_results
Information taken from l3fCalMEt_phys_results and l3fJetMEt_phys_results
L3MEtPhysicsResults.hpp

Accessor	*Method used*		
Phi()	get_MEtPhi()	float	azimuthal angle

ToolName()	l3mapIter->first	string	trigger list name of the physics tool that produced the candidate
MEt()	get_MEt()	float	missing transverse energy
MEx()	get_MEx()	float	x component of missing transverse energy
MEy()	get_MEy()	float	y component of missing transverse energy
ScalarEt()	get_ScalarET()	float	scalar sum of the transverse energy
MEtsignif()	get_MEtSignif()	float	significance of the missing transverse energy (MEt/$\sqrt{\text{ScalarEt}}$)
AlgFlag()	get_AlgFlag()	int	algorithm = 1000*(Muon Corr.?) + 100*(Towers?) + 10*(Cells?) + 1*(UseCH?) (not filled in the case of L3Ht)

A.6.5 L3Taus

The L3Taus branch is a TClonesArray of L3 tau candidates.

Header file: L3Tau.hpp
Package used to fill the chunk read : l3fTauTools_physres
Information taken from
http://www-d0.fnal.gov/computing/trigsim/general/docs/tuple-info/L3Tau.html and DØ Note 4132

Accessor	*Method used*		
Et()	get_ET()	float	cluster transverse energy
Eta()	get_detectorEta()	float	cluster pseudorapidity
Phi()	get_kineResults()->phi()	float	cluster azimuthal angle
ToolName()	l3mapIter->first	string	trigger list name of the physics tool that produced the candidate
Iso()	get_isolation()	float	cluster isolation
Emf()	get_emFraction()	float	cluster EM fraction
Charge()	get_charge()	int	charge
SeedAlgo()	get_seed_algo()	int	algorithm used (1 = CAL_BASED, 2 = TRACK_BASED, 3 = NN_BASED)
Width()	get_width()	float	transverse energy-weighted cluster width
Profile()	get_profile()	float	ratio of the sum of the transverse energy of the two highest transverse energy towers in the cluster to the total cluster transverse energy
EM12isof()	get_EM12isof()	float	profile for the cluster towers contained in the isolation cone of 0.5 only
NNOut()	get_NN_Out()	double	neural network output value
NTracks()	get_ntracks()	int	number of tracks pointing to the cal cluster if CAL_BASED, or number of tracks in track cluster if TRACK_BASED
EM3Eta()	get_em3_eta()	float	cluster pseudorapidity in EM3 layer
EM3Phi()	get_em3_phi()	float	cluster azimuthal angle in EM3 layer
EM3E()	get_em3_E()	float	cluster energy in EM3 layer

M01()	get_m01()	float	if TRACK_BASED, invariant mass of first 2 tracks
M012()	get_m012()	float	if TRACK_BASED, invariant mass of first 3 tracks
SumPt3()	get_sumpt3()	float	if TRACK_BASED and more than 3 matching tracks, sum of $\|p_T\|$ of those extra tracks

A.6.6 L3Muons

The L3Muons branch is a TClonesArray of L3 muon candidates.

Header file: trigsimcert/L3Muon.hpp
Package used to fill the chunk read: l3fMuonTools_results
Information taken from l3fmuo_local/L3MuoTrack.hpp, l3fMuonTools_results/L3MuonPhysicsResults.hpp and DØ Note 4091

Accessor	*Method used*		
Et()	get_ET()	float	tranverse energy
Eta()	get_eta()	float	p_T weighted pseudorapidity
Phi()	get_phi()	float	p_T weighted azimuthal angle
ToolName()	l3mapIter->first	string	trigger list name of the physics tool that produced the candidate
Overall muon info			
Region()	get_region()	int	region (0: central, 1: North $[z < 0]$, 2: South $[z > 0]$)
Octant()	get_octant()	int	position octant
Pt()	get_pT()	float	tranverse momentum
Z()	get_z()	float	z coordinate
Charge()	get_charge()	int	charge
Local muon info			
EtaLocal()	get_etaLocal()	float	pseudorapidity in the A-layer
PhiLocal()	get_phiLocal()	float	azimuthal angle in the A-layer
PtLocal()	get_pTLocal()	float	transverse momentum of the local track
ZLocal()	get_zLocal()	float	z coordinate of the local track
Quality()	get_quality()	int	quality of the local track (L3MUNONE = 0, L3MUASTUB = 1, L3MULOOSE = 2, L3MUMEDIUM = 3, L3MUTIGHT = 4)
From calorimeter			
CalMatched()	isCalmatched()	bool	matched to a calorimeter cluster
CalMatchedRetval()	get_calmatchReturnInt()	int	result returned by the calorimeter match
MTCEtrack()	get_MTC_Etrack()	float	track energy in the calorimeter

MTCHfrac()	get_MTC_Hfrac()	float	fraction of hit cells of muon in the hadronic part of the calorimeter
From central tracker			
CentralMatched()	isCentralmatched()	bool	matched to a central track
CentralMatchRetval()	get_centralmatchReturnInt()	int	result returned by the central match
ChisqCentral()	get_chisqCentral()	float	χ^2 of the central track fit
EtaCentral()	get_etaCentral()	float	pseudorapidity of the central track
PhiCentral()	get_phiCentral()	float	azimuthal angle of the central track
PtCentral()	get_pTCentral()	float	transverse momentum of the central track
ZCentral()	get_zCentral()	float	z coordinate of the central track
ImpactXY()	get_impactXY()	float	impact parameter of the central track
SignifImpactXY()	get_signifImpactXY()	float	Impact paramemter significance

A.6.7 L3BTagIP

The L3BTagIP branch is a TClonesArray of L3 b tagging results.

Header file: trigsimcert/L3BTagIP.hpp
Package used to fill the chunk read: l3fbtag_ip_results
Information taken from l3fbtag_ip_result/src/L3BTagIPPhysicsResults.cpp and
http://www-d0.fnal.gov/computing/algorithms/level3/b-tagging/L3Btag.html

Accessor	*Method used*		
ToolName()	l3mapIter->first	string	trigger list name of the physics tool that produced the candidate
BTag()	get_btag()	float	probability of the presence of a b quark in the event based on the signed impact parameter significance of tracks belonging to the leading jets.

A.6.8 L3Isolation

The L3Isolation branch is a TClonesArray of L3 isolation results.

Header file: trigsimcert/L3Isolation.hpp
Package used to fill the chunk read: l3fisolation_results
Information taken from l3fisolation/L3TIsolation.hpp

Accessor	*Method used*		
ToolName()	l3mapIter->first	string	trigger list name of the physics tool that produced the candidate
ConeEnergy()	getConeEnergy()	float	energy in hollow cone
TrackIsolation()	getTrackIsolation()	int	track is isolated from the other tracks

TrackPtSum()	getTrackptSum()	float	p_T sum of swarm tracks inside the cone
Distances()	getDistances()	float	distances of swarm tracks to active track

A.6.9 L3IPTracks

The L3IPTracks branch is a TClonesArray of L3 tracks found by an impact parameter-based algorithm. It returns the primary vertex information as well as track fit information for each entry.

Header file: trigsimcert/L3IPTrack.hpp
Package used to fill the chunk read: l3fip_track_results
Information taken from l3fip_track_results/L3IPTrackPhysicsResults.hpp, l3fip_track/src/L3TIPTracker.cpp and l3ftrack_global/L3TGlobalTracker.hpp

Accessor	*Method used*		
ToolName()	l3mapIter->first	string	trigger list name of the physics tool that produced the candidate
VertexX()	get_VertexX()	double	x coordinate of the vertex
VertexErrX()	get_VertexErrX()	double	error on the x coordinate of the vertex
VertexY()	get_VertexY()	double	y coordinate of the vertex
VertexErrY()	get_VertexErrY()	double	error on the y coordinate of the vertex
VertexZ()	get_VertexZ()	double	z coordinate of the vertex
VertexErrZ()	get_VertexErrZ()	double	error on the z coordinate of the vertex
TrackRINVUncorr()	get_TrackRINV_UNCORR()	double	uncorrelated inverse radius of the track
TrackPTINVUncorr()	get_TrackPTINV_UNCORR()	double	uncorrelated inverse transverse momentum of the track
TrackTANLUncorr()	get_TrackTANL_UNCORR()	double	uncorrelated $\tan\lambda$ of the track
TrackPHIUncorr()	get_TrackPHI_UNCORR()	double	uncorrelated azimuthal angle of the track
TrackDCAUncorr()	get_TrackDCA_UNCORR()	double	uncorrelated distance of closest approach of the track
TrackZ0Uncorr()	get_TrackZ0_UNCORR()	double	uncorrelated z of the track
TrackDCAErrUncorr()	get_TrackDCAERR_UNCORR()	double	uncorrelated error on the distance of closest approach of the track
TrackSIGUncorr()	get_TrackSIG_UNCORR()	double	uncorrelated significance of the track fit
TrackRINVCorr()	get_TrackRINV_CORR()	double	correlated inverse radius of the track
TrackPTINVCorr()	get_TrackPTINV_CORR()	double	correlated inverse transverse momentum of the track

TrackTANLCorr()	get_TrackTANL_CORR()	double	correlated $\tan\lambda$ of the track
TrackPHICorr()	get_TrackPHI_CORR()	double	correlated azimuthal angle of the track
TrackDCACorr()	get_TrackDCA_CORR()	double	correlated distance of closest approach of the track
TrackZ0Corr()	get_TrackZ0_CORR()	double	correlated z of the track
TrackDCAErrCorr()	get_TrackDCAERR_CORR()	double	correlated distance of closest approach error of the track
TrackSIGCorr()	get_TrackSIG_CORR()	double	correlated significance of the track fit
TrackNSMTHitsXY()	get_TrackNSMTHitsXY()	int	number of hits in the axial layers of the SMT
TrackNCFTHitsXY()	get_TrackNCFTHitsXY()	int	number of hits in the axial layers of the CFT
TrackNSMTHitsZ()	get_TrackNSMTHitsZ()	int	number of hits in the stereo layers of the SMT
TrackNCFTHitsZ()	get_TrackNCFTHitsZ()	int	number of hits in the stereo layers of the CFT

A.6.10 L3Tracks

The L3Tracks branch is a TClonesArray of L3 tracks. All methods in the *Method used* column of this table have to be preceded by get_Track(). For example, nHitsZ() is actually get_Track().nHitsZ().

Header file: trigsimcert/L3Trk.hppx
Package used to fill the chunk read: l3ftrack_phys_results
Information taken from l3ftrack_base/l3ftrack_base/L3TrackFit.hpp

Accessor	*Method used*		
Et()		float	transverse momentum (|1./PtInv()|)
Eta()	*from* getParam(L3TrackParams::TANL)	float	pseudorapidity
Phi()	getParam(L3TrackParams::PHI)	float	azimuthal angle
ToolName()	l3mapIter->first	string	trigger list name of the physics tool that produced the candidate
PtInv()	getParam(L3TrackParams::PTINV)	double	inverse transverse momentum
Z()	getParam(L3TrackParams::Z)	double	z coordinate (in cm)
DCA()	getParam(L3TrackParams::DCA)	double	distance of closest approach
tanl()		float	$\tan\lambda$, derived from Eta()
ChiSq()		float	Sum of ChiSqXY() and ChiSqZ()
ChiSqXY()	getChiSqXY()	float	axial fit χ^2
ChiSqZ()	getChiSqZ()	float	stereo fit χ^2

ValidFit()		bool	both axial and stereo fits were successful (derived from ValidXYFit() and ValidZFit())
ValidXYFit()	isValidXY()	bool	the axial fit was successful
ValidZFit()	isValidZ()	bool	the stereo fit was successful
nHitsZ()	nHitsZ()	int	number of hits in stereo layers
nHitsXY()	nHitsXY()	int	number of hits in axial layers
nSMTHitsZ()	nSMTHitsZ()	int	number of hits in SMT stereo layers
nSMTHitsXY()	nSMTHitsXY()	int	number of hits in SMT axial layers

A.6.11 L3CFTVertex

The L3CFTVertex branch is a TClonesArray of L3 vertexing results in the CFT.

Header file: trigsimcert/L3CFTVtx.hpp
Package used to fill the chunk read: l3fcft_vertex_phys_results
Information taken from l3fcft_vertex/src/L3TCFTVertex.cpp

Accessor	*Method used*		
ToolName()	l3mapIter->first	string	trigger list name of the physics tool that produced the candidate
X()	x()	float	x coordinate
Y()	y()	float	y coordinate
Z()	z()	float	z coordinate
ErrX()	errx()	float	error on x
ErrY()	erry()	float	error on y
ErrZ()	errz()	float	error on z
Weight()	weight()	float	weight of the vertex
Primary()	is_primary()	bool	the vertex is a primary vertex

B Example of a Level 1 Histogram Class

```
////////////////////////////////////////////////////////////////////////////
//
//    File: Hl1muon.C
//    Author : Steve Beale / Yann Coadou (SFU) for generalisation
//    Purpose : histogram class for the L1Muon branch.
//    Created : 9 May 2004
//
////////////////////////////////////////////////////////////////////////////

#ifndef L1MUON_H_
#include "trigsimcert/L1Muon.hpp"
#endif
#include "headers.h"
#ifndef HDRAW_H_
#include "HDraw.h" //The histogram drawing class, to be inherited.
#endif

////////////////////////////////////////////////////////////////////////////
//                              IMPORTANT!!!!                            //
// If you change the histogram properties you must regenerate the         //
// reference file 'Ref.root': To do this make a 'Plots' object with the   //
// reference file, and then run Loop(true) ie:                            //
//                                                                        //
// .x macros/LoadPlots.C                                                  //
// Plots anyname("input/file","output/path/dir/");                        //
// anyname.Loop(true);                                                    //
//                                                                        //
// Where 'input/file'  is a text file containing the path to the desired  //
// reference root tuple, and Ref.root will be placed in                   //
// 'output/path/dir/Ref.root'                                             //
// IF YOU DO NOT REGENERATE THE REFERENCE FILE THE PLOTS WILL NOT WORK    //
////////////////////////////////////////////////////////////////////////////

class Hl1muon : public HDraw
{
  public :
    Hl1muon();
    ~Hl1muon();
    void Fill(L1Muon *l2obj);
    void DoneEvent();
    //Takes a reference file as input and ps file for output
    void Draw(TFile *ref,TPostScript *ps,bool debug=false);
    Bool_t Save(TFile *file); //returns true for successful save
```

```
  private :
    Float_t HISTO1max;
    Float_t HISTO1min;
    Int_t   HISTO1bins;
    std::string HISTO1axis;
    Float_t HISTO2max;
    Float_t HISTO2min;
    Int_t   HISTO2bins;
    std::string HISTO2axis;
    Float_t HISTO3max;
    Float_t HISTO3min;
    Int_t   HISTO3bins;
    std::string HISTO3axis;

    Int_t Nresults; //Number of objects/event

    //histogram pointers:
    TH1F *HISTO1;
    TH1F *HISTO2;
    TH1F *HISTO3;

    char *THISFILE;
};

Hl1muon::Hl1muon()
{
  HISTO1max=256;
  HISTO1min=0;
  HISTO1bins=256;
  HISTO1axis="# central octants";
  HISTO2max=256;
  HISTO2min=0;
  HISTO2bins=256;
  HISTO2axis="# North endcap octants";
  HISTO3max=256;
  HISTO3min=0;
  HISTO3bins=256;
  HISTO3axis="# South endcap octants";

  Nresults = 0;
  //make histograms:
  HISTO1=new TH1F("l1muon_HISTO1","L1Muon number of central octants",
  HISTO1bins,HISTO1min,HISTO1max);
  HISTO2=new TH1F("l1muon_HISTO2","L1Muon number of North endcap octants",
  HISTO2bins,HISTO2min,HISTO2max);
  HISTO3=new TH1F("l1muon_HISTO3","L1Muon number of South endcap octants",
```

```
  HISTO3bins,HISTO3min,HISTO3max);

  //Set X axis labels:
  HISTO1->GetXaxis()->SetTitle(HISTO1axis.c_str());
  HISTO1->GetXaxis()->CenterTitle();
  HISTO2->GetXaxis()->SetTitle(HISTO2axis.c_str());
  HISTO2->GetXaxis()->CenterTitle();
  HISTO3->GetXaxis()->SetTitle(HISTO3axis.c_str());
  HISTO3->GetXaxis()->CenterTitle();
  //Just set THISFILE
  THISFILE = "Hl1muon.C";
  std::cout<<"Initializing "<<THISFILE<<std::endl;

}

Hl1muon::~Hl1muon()
{
  delete HISTO1;
  delete HISTO2;
  delete HISTO3;

  std::cout<<"Cleared l1muon histograms"<<std::endl;
}

void Hl1muon::Fill(L1Muon *l1obj)
{
  HISTO1->Fill(l1obj->nCenTrigOct());
  HISTO2->Fill(l1obj->nNorthTrigOct());
  HISTO3->Fill(l1obj->nSouthTrigOct());
  ++Nresults;
}

void Hl1muon::DoneEvent()
{
  Nresults=0;
}

Bool_t Hl1muon::Save(TFile *file)
{
  if(file!=0)
    {
      file->cd();
      HISTO1->Write();
      HISTO2->Write();
      HISTO3->Write();
      return true;
```

```
    }
  else return false;
}

////////////////////////////////////////////////////////////////////////
//                               Draw                                 //
////////////////////////////////////////////////////////////////////////

void Hl1muon::Draw(TFile *ref,TPostScript *ps,bool debug)
{
  bool drawref=true;
  bool post=true;
  if(ref==0) //then do not draw reference
    { drawref=false; }
  if(ps==0) //then draw to screen not postscript
    { post=false; }

  //Draw stuff
  if(post) ps->Off(); //deactivate ps until ready
  TCanvas *cl1muon = new TCanvas("cl1muon","cl1muon");
  cl1muon->Divide(2,2);

  //////////////////////HISTO1///////////////////////////////

  cl1muon->cd(1);
  TLegend *Leg1=new TLegend(.75,.75,.99,.99,"");
  if(drawref)
    {
      TH1F *HISTO1ref = (TH1F*) ref->Get("l1muon_HISTO1");
      DrawHisto(HISTO1,drawref,HISTO1ref,Leg1,"HISTO1",THISFILE);
    }
  else
    DrawHisto(HISTO1,drawref,0,Leg1,"HISTO1",THISFILE);

  //////////////////////HISTO2///////////////////////////////

  cl1muon->cd(2);
  TLegend *Leg2=new TLegend(.75,.75,.99,.99,"");
  if(drawref)
    {
      TH1F *HISTO2ref = (TH1F*) ref->Get("l1muon_HISTO2");
      DrawHisto(HISTO2,drawref,HISTO2ref,Leg2,"HISTO2",THISFILE);
    }
  else
    DrawHisto(HISTO2,drawref,0,Leg2,"HISTO2",THISFILE);
```

```
  ///////////////////////HISTO3//////////////////////////////////

  cl1muon->cd(3);
  TLegend *Leg3=new TLegend(.75,.75,.99,.99,"");
  if(drawref)
    {
      TH1F *HISTO3ref = (TH1F*) ref->Get("l1muon_HISTO3");
      DrawHisto(HISTO3,drawref,HISTO3ref,Leg3,"HISTO3",THISFILE);
    }
  else
    DrawHisto(HISTO3,drawref,0,Leg3,"HISTO3",THISFILE);

  if(post){ ps->On();  ps->NewPage();}//reactivate ps and make a new page
  cl1muon->Update(); if(post){ ps->Off(); }
  if(!debug) cl1muon->Close(); //Keep canvas open if debugging
}
```

C Macros Usage Examples

C.1 Your first trigsimcert session

Let us assume that trigsimcert was run on a file produced by p16 trigsim and then on a file produced by p17 trigsim. The respective trigsimcert ROOT tree file names were put in text files p16trigcertList and p17trigcertList in your working directory. Assume also there is a subdirectory there called MyComparisons. You can now go to your working directory and do the following:

```
setup D0RunII p17.00.00
d0setwa
root -b
```

Starting ROOT with the -b option prevents canvases created by the macros from popping up. In ROOT, start by loading and compiling the macros:

```
root [0] .x trigsimcert/macros/LoadPlots.C
```

The first time is a bit slow, but the consecutive times will just load the precompiled macros, unless you have modified them. You can now create a `Plots` object with the desired input (the p16trigcertList file list) and output (the MyComparisons directory) parameters:

```
root [1] Plots YourPlot("p16trigcertList","MyComparisons");
```

In order to produce a reference histogram set out of this p16 tree, the `Loop` method of the `Plots` object should be called with its argument set to true:

```
root [2] YourPlot.Loop(true);
```

This produces a file called Ref.root in MyComparisons, containing all reference histograms. For some obscure reason known only to ROOT, you should now quit and restart ROOT, and reload the classes:

```
root [0] .x trigsimcert/macros/LoadPlots.C
```

Now a new `Plots` object is instantiated and its `Loop` method is called, without an argument (by default it does not make a reference) to produce comparison plots between p16 and p17:

```
root [1] Plots YourPlot("p17trigcertList","MyComparisons");
root [2] YourPlot.Loop();
```

Now in the MyComparisons directory you will find a ROOT file, plots.root, similar to the Ref.root file, containing all histograms from p17. There is also a plots.ps file that contains comparisons between p16 and p17.

The same result can be achieved using the analysis executable, as described in section C.2.

C.2 Your first trigsimcert session using the executable

The same result as what was obtained in section C.1 can be achieved by making an executable out of the macros. The same configuration is assumed.

The first step is to compile the executable:

```
cd trigsimcert/macros
make
```

After a few seconds it should produce an executable and give the most important information:

```
./TrigCertAna -h to see how to use the program
```

In order to achieve the same results as described in section C.1 one can now execute the following to produce the reference histograms:

```
./TrigCertAna -filelist p16trigcertList -name MyComparisons -makeref
```

Another very similar command line will now produce the comparison plots (just drop the -makeref argument):

```
./TrigCertAna -filelist p17trigcertList -name MyComparisons
```

C.3 Using trigsimcert for a trigger analysis

The trigsimcert macros can be used as the basis for a trigger analysis. Most trigger objects are accessed by the Plots.C macro, which can be used as a template. Modifying this file, one can then use the machinery in place to run, compile or produce comparison plots, within ROOT or using the Makefile. The macros provide examples of how to access the information and it is easy to remove anything unnecessary.

As an example of what is probably of interest for any trigger analyser, here is how one would run the macros only on events that passed a specific trigger. This code would go in the `Plots` method of the Plots.C file:

```
bool passed = false;
string MyTrigger = "E1_SHT22";
for (int j = 0; j < (ftrigger->GetLast()+1); ++j) {
  trigger = dynamic_cast<Trigger*>(ftrigger->At(j));
  if (trigger->L3Name() == MyTrigger && trigger->L3Passed()) passed = true;
}
if (passed) { Do my analysis }
```