`HWWAnalysisCode`

Tutorial Sessions

Session 2/2

Running on ntuples

Andreas Walz
andreas.walz@cern.ch

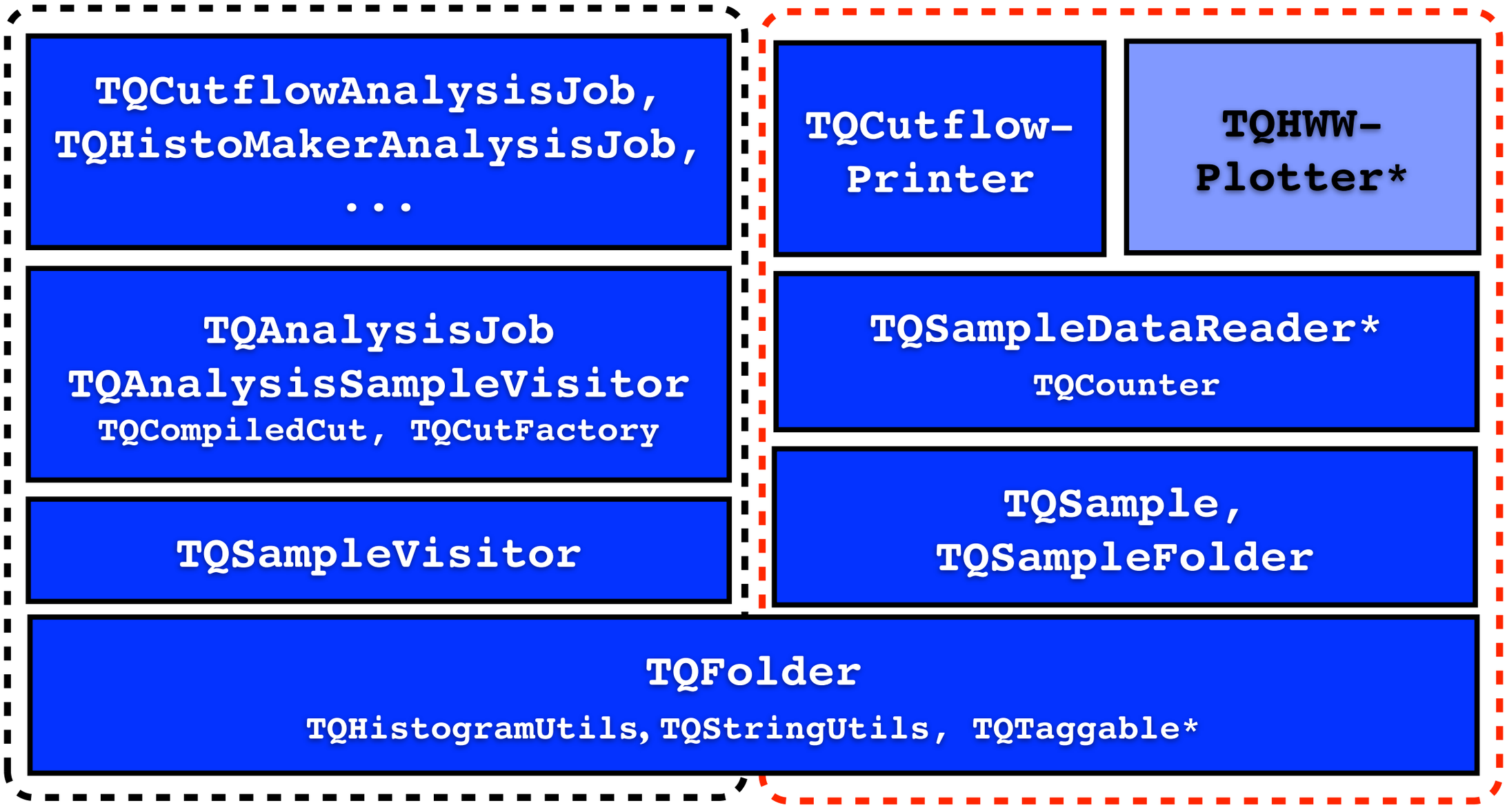
Albert-Ludwigs-Universität Freiburg

2012-05-11

Recap: Structure of the library

Running analysis (producing analysis results, histograms, ...)

Management and presentation of analysis results (histograms, ...)



Addition to session 1

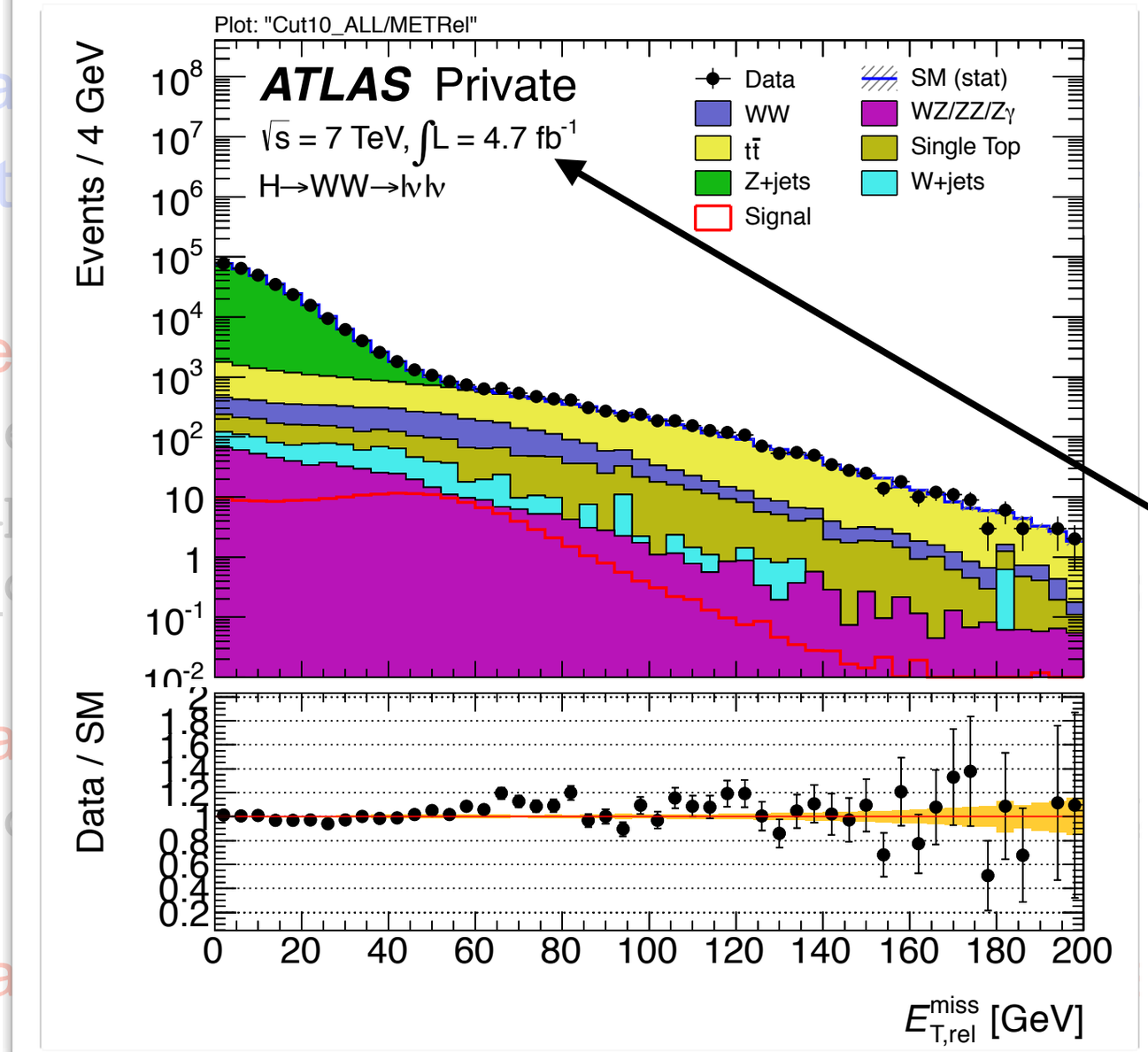
There was an important method missing in SVN on Tuesday to make the `TQHWWPlotter2` class accepting options.

```
// load the sample folder of exercise 5 (session 1)
TQSampleFolder * samples =
  TQSampleFolder::loadSampleFolder(
  „hww_dataMC_genHisto.root:samples“);
```

```
// create an instance of the HWW plotter 2:
TQHWWPlotter2 * pl = new TQHWWPlotter2(samples);
```

```
// create a plot with data/MC ratio, label and in log scale
TCanvas * c = pl->plot(
  "Cut10_ALL/METRel", "style.showRatio=true,
  labels.lumi='#sqrt{s} = 7 TeV, #scale[0.7]
  {#int}L = 4.7 fb^{-1}', style.logScale=true");
```

Addition to session 1



lumi label

```
"Cut10_ALL/METRel", "style.showRatio=true,
labels.lumi='#sqrt{s} = 7 TeV, #scale[0.7]
{#int}L = 4.7 fb^{-1}', style.logScale=true");
```

How to set up an analysis (in general)

The first step to run an analysis using the library of the `HWWAnalysisCode` is to set up the sample folder hierarchy representing the analysis

1) create the root sample folder (an instance of `TQSampleFolder`)

2) create the sub sample folders (instances of `TQSampleFolder`) categorizing the samples

3) create samples (instances of `TQSample`) representing an atomic category of event samples (technically a sample taken from a single `ROOT` tree)

1) Create the root sample folder

Creating an empty sample folder (automatically being the root sample folder):

not necessarily the same

```
TQSampleFolder * samples =
  TQSampleFolder::newSampleFolder(„samples“);
```

There are constraints about the names that are allowed:

- small and capital letters (**`a`** to **`z`**, **`A`** to **`Z`**)
- numerals (**`0`** to **`9`**)
- dot (**`.`**), underscore (**`_`**)

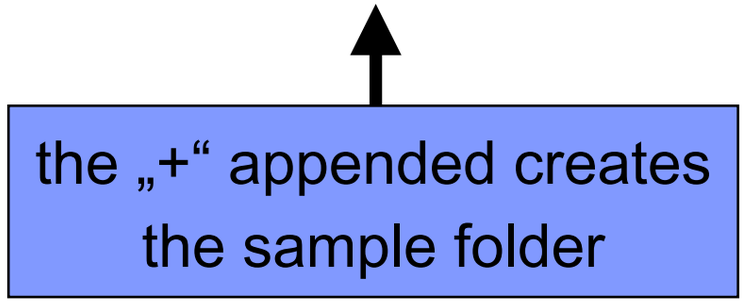
Examples

- allowed: „`samples`“, „`myFolder_123`“, „`my.folder`“
- not allowed: „`my folder`“, „`my-folder`“

2) Create the sub sample folders

Starting from the empty root sample folder, create the sample folder hierarchy categorizing your samples, e.g.:

```
samples->getSampleFolder(„bkg/ttbar+“);
samples->getSampleFolder(„bkg/Zjets+“);
samples->getSampleFolder(„data/period_A+“);
...
```



The structure of the sample folder hierarchy is completely arbitrary and can be choosen according to any analysis!

3) Create the samples

Given the sample folder hierarchy created in the previous step, create the samples as sub elements of the sample folders, e.g.:

```
TQSample * ttbar_105200 =
  new TQSample(„105200“);
samples->getSampleFolder(„bkg/ttbar“)
  ->addSampleFolder(ttbar_105200);
```

or merging step 2) and 3)

```
samples->addSampleFolder(
  new TQSample(„105200“), „bkg/ttbar+“);
```

The `TQSample` class is a specialization of the `TQSampleFolder` class, that‘s why you use `addSampleFolder()` to add a sample.

H→WW→lνlν specific

Setting up the H→WW→lνlν analysis

In the default H→WW→lνlν analysis the sample folder hierarchy is created from a cross section file using the `TQHWWXSecParser` class infering the categorization of a sample from a process info string in the file:

```
parser->readXSecFile(
   "Xsection_bkg.txt", samples, 1);
```

- the file to parse
- the sample folder to create the new hierarchy in
- minimal (numerically the maximal) sample priority

```
##Format                                                                                    ##
DatasetID : Xsection(pb) : K-factor : Filtering Efficiency : Mass : Sample Priority : Generator : ProcessInfo ##
##############################################################################################

105921      0.520       1.0      1.0           -999      1      MC@NLO                qq->WpWm->eenunu
105922      0.520       1.0      1.0           -999      1      MC@NLO                qq->WpWm->emununu
105923      0.520       1.0      1.0           -999      1      MC@NLO                qq->WpWm->etaunu nu
```

H→WW→lνlν specific

Example 1: TQHWWXSecParser

```
// create an empty sample folder
TQSampleFolder * samples =
  TQSampleFolder::newSampleFolder(„samples“);

// create an instance of the parser
TQHWWXSecParser * parser =
  new TQHWWXSecParser();
```

the parser automatically creates three samples for every entry in the cross section file (for ee, eμ and μμ final state respectively)

```
// parse the cross section file for background samples
parser->readXSecFile(
  „../XsectionInput/Xsection_bkg_v5.txt“,
  samples, 1);

// print the hierarchy and some tags
samples->print(„rd“);
samples->getSample(„*/105921“)->printTags();
```

H→WW→lνlν specific

Exerc. 11: Customized sample folders

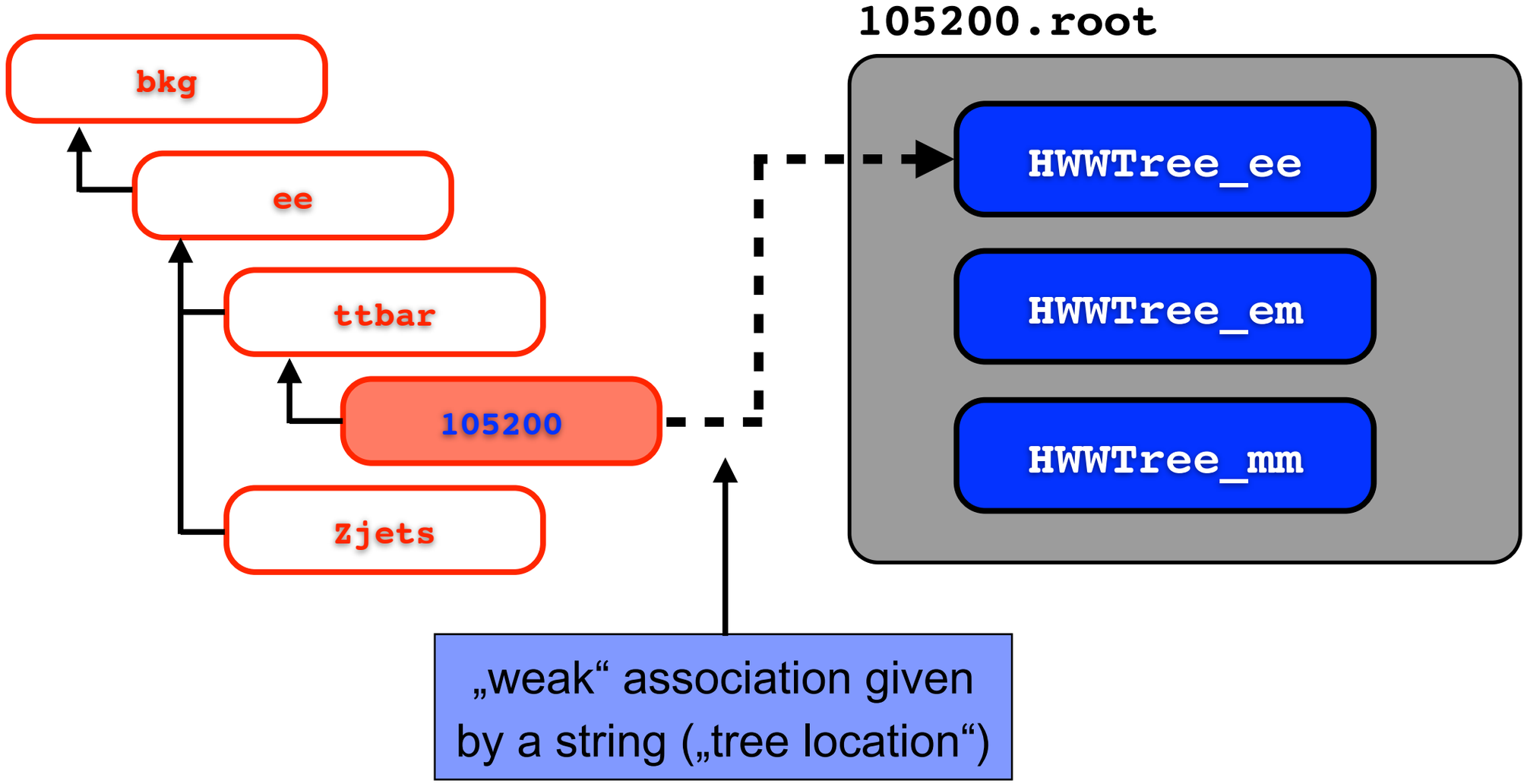
Use the `TQHWWXSecParser` class to create two instances of the default H→WW→lνlν background sample folder hierarchy in two different sub sample folders (e.g. „`atlfast`“ and „`fullsim`“) of one common root sample folder.

Some comments/hints:

- use „`../XsectionInput/Xsection_bkg_v5.txt`“ as cross section input file
- use **`gROOT->Add(samples);`** at the end of your macro to browse the hierarchy in the `ROOT` command line even after your macro has terminated

The `TQSample` class

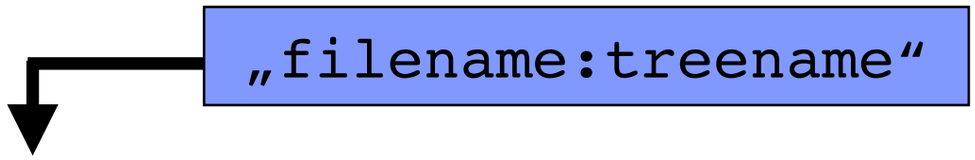
The `TQSample` class is a specialization of the `TQSample-Folder` class representing a leaf of the sample folder hierarchy. Being an „atomic“ category of event samples, it has an association to a single `ROOT TTree` object („ntuple“)



continued: The `TQSample` class

The `TQSample` class introduces some additional (with respect to the `TQSampleFolder` class) features and parameters:

- the association to a `TTree` („tree location“, see previous slide):



  - **`setTreeLocation(„treeLocation“)`**
  - **`getTreeLocation()`**

- a normalization factor being applied to every histogram and cutflow counter (before being stored)

  - **`setNormalisation(...)`**
  - **`getNormalisation()`**

Accessing the `TTree` of a sample

The `TQSample` class governs the access to the `TTree` object associated to the sample, including opening/closing the `ROOT` file containing the tree. The file access is organized by dispensing tree tokens to the user:

- **`TQToken * treeToken = sample->getTreeToken();`**

The first tree token request will trigger the opening of the file

```
TTree * tree =
  (TTree*)treeToken->getContent();
```

Return the tree token when you are done:

- **`sample->returnTreeToken(treeToken);`**

Returning the last tree token will trigger the closing of the file

Example 2: `TQSample` class

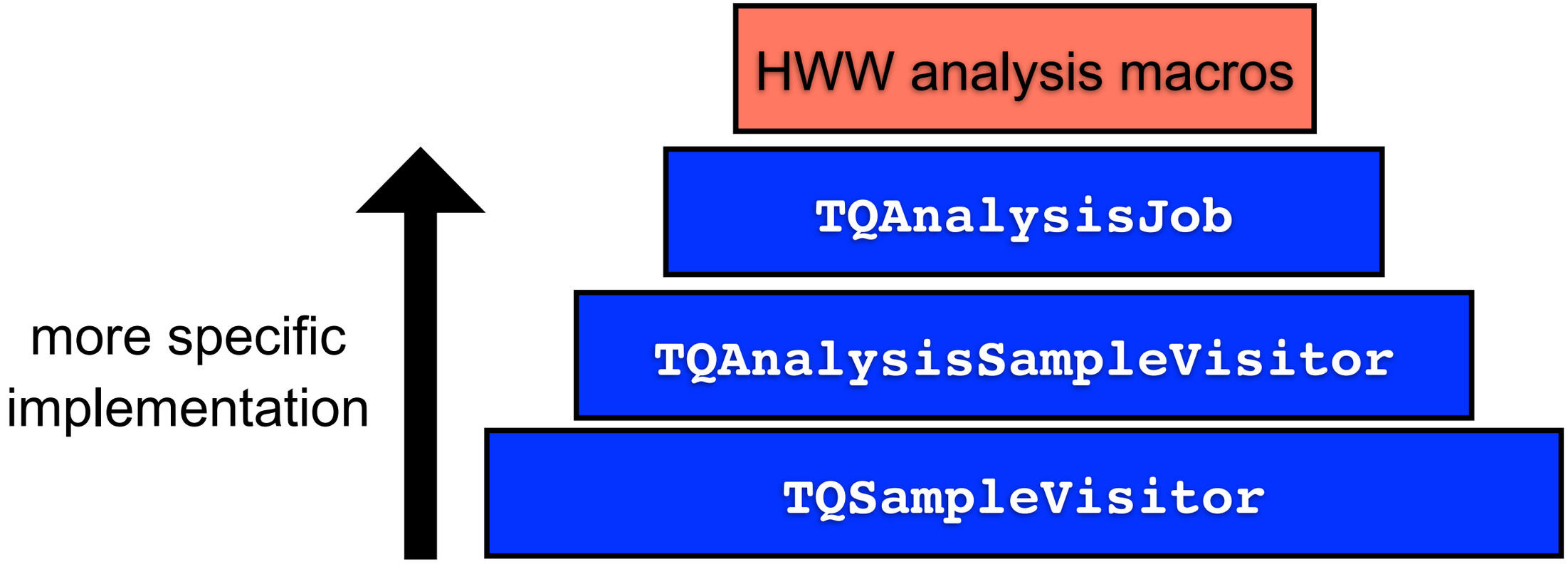
```
// create a new instance of the TQSample class
TQSample * sample = new TQSample(„mySample“);

// set the sample‘s normalization factor (e.g. 10-3)
sample->setNormalisation(1E-3);

// set the sample‘s tree location
sample->setTreeLocation(„/afs/cern.ch/work/a/“
   „awalz/public/ntuples/105200.root:HWWTree_ee“);

// access the tree (requesting a tree token)
TQToken * treeToken = sample->getTreeToken();
TTree * tree = (TTree*)treeToken->getContent();
tree->Scan(„lepPt0:lepPt1“);

// return the tree token
sample->returnTreeToken(treeToken);
```

just for illustration, does not affect this example

Analysis „working stack“

The library provides a stack of classes designed to run on samples („ntuples“), each using the services provided by the classes of the levels below.

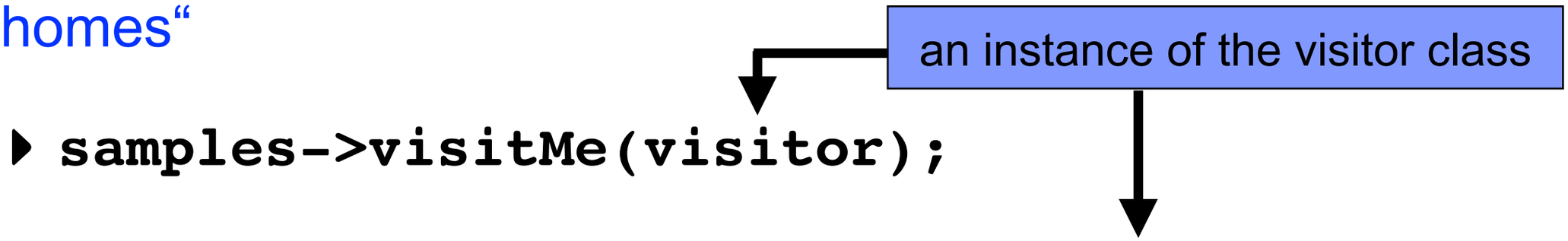
Going to higher levels, the implementation gets more specific



The „visitor pattern“

„Heavy computational jobs“ to be performed on the sample folder hierarchy are encapsulated in visitor classes inheriting from the `TQSampleVisitor` class.

„Instead of bringing cars to the car workshop, let a service person go around maintaining (visiting) the cars at their homes“



- **`samples->visitMe(visitor);`**
- **`samples->visitSampleFolders(visitor, „bkg/ee, sig/ee/mh125, data/ee“);`**

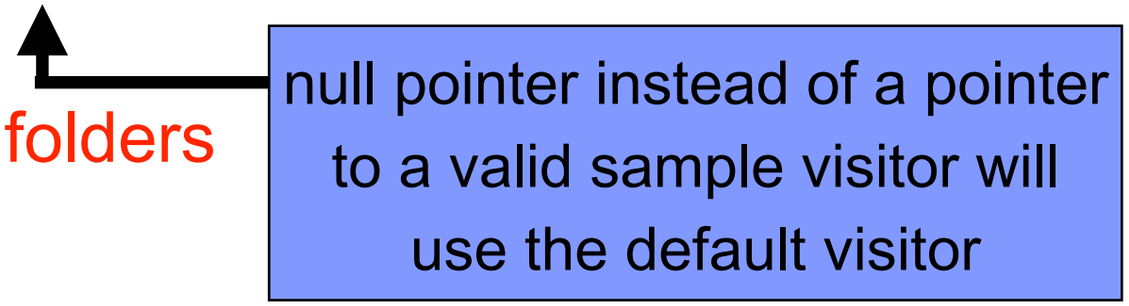
Examples of pre-implemented visitor classes:

- `TQAnalysisSampleVisitor`,
  `TQHWWSampleInitializer`

Example 3: Default visitor

The `TQSampleVisitor` class provides a default implementation of a simple visitor job (listing samples)

```
// load the example sample folder from the external ROOT file
TQSampleFolder * samples =
  TQSampleFolder::loadSampleFolder(
  „example3.root:samples“);

// run the default visitor on background sample folders
samples->visitSampleFolders(0, „bkg/ee“);

// run the default visitor on all sample folders
samples->visitMe();
```



H→WW→lνlν specific

The `TQHWWSampleInitializer`

The `TQHWWSampleInitializer` class is used to initialize the `TQSample` objects (representing Monte Carlo samples) in the sample folder hierarchy by visiting these samples.

The initializer sets:

- the normalization factor using the predefined (integrated) luminosity to normalize the MC to, cross section and filter efficiency of the sample and the number of events generated (taken from the `Count` histogram in the ntuple)
- the tree location using the dataset id of the sample and a predefined file path

H→WW→lνlν specific

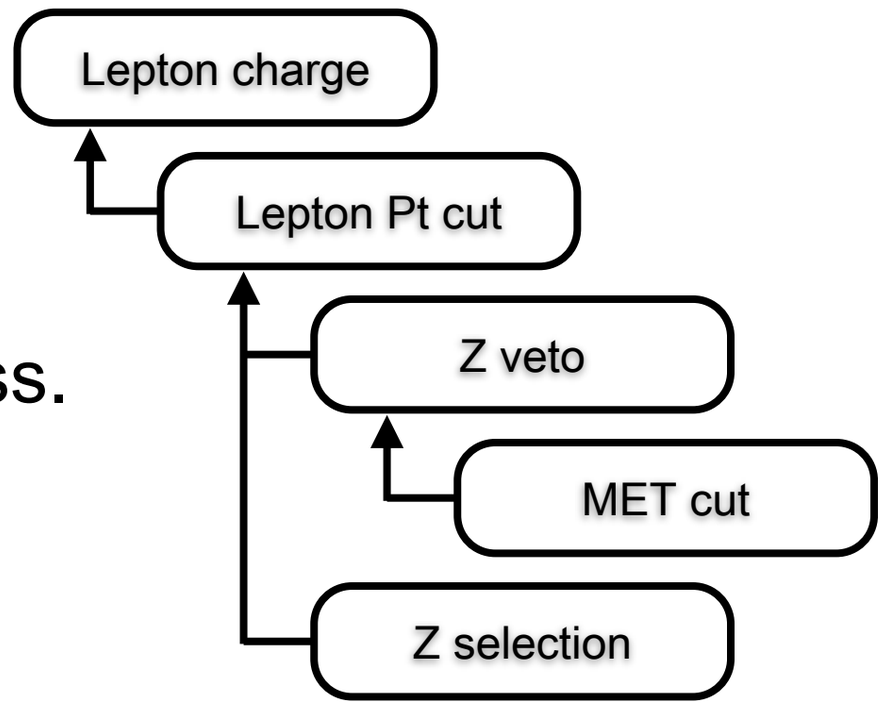
Ex. 4: TQHWWSampleInitializer

```
// ... get sample folder hierarchy of Example 1
...

// create an instance of the sample initializer
TQHWWSampleInitializer * initializer =
  new TQHWWSampleInitializer();

// set initializer parameters
initializer->setLuminosity(4712.);
initializer->setFilepath(„/afs/cern.ch/work/a/“
  „awalz/public/ntuples“);
initializer->setNEventsBin(1);

// run the initializer
samples->visitMe(initializer);
```

Modeling event selection cuts

Event selection cuts can be organized in a tree-like hiearchy with the „base cut“ being the precondition for the cut under consideration.

In the `HWWAnalysisCode` event selection cuts are represented by instances of the `TQCompiledCut` class.



Cut hierarchy can be printed with

- **`cut->print();`**

```
Name                          # Jobs   Cut Expression
==========================================================================================
Cut6                               0   1.
  Cut7                             0   isLowPtCand == 0. && (lepPt0 > 25000. || lepPt1 > 25000.)
    Cut8                           0   lepID0*lepID1 < 0.
      Cut9                         0   Mll > 12000.
        Cut10                      0   abs(Mll - 91187.6) > 15000.
          Cut11                    0   METRel > 45000.
```

The `TQCutFactory` class

The `TQCutFactory` class is used to compile a tree of `TQCompiledCut` objectes from a simple cut definition syntax:

- **`factory->addCut(„cut definition“);`**
  - add a cut to the definition of the cut tree

optional

```
„cutName: preConditionCut <<
cutExpression : weightExpression“
```

- **`factory->compileCuts(„parameter“);`**
  - compile the cut hierarchy and return a pointer to the `TQCompiledCut` object representing the root selection cut

Example 5: TQCutFactory

```
// create an instance of the cut factory
TQCutFactory * cf = new TQCutFactory();

// define two cuts
cf->addCut(„rootCut: nLeptons == 2“);
cf->addCut(„leptonCut: rootCut <<
  lepCharge0 != lepCharge1“);

// compile the cut hierarchy
TQCompiledCut * cuts = cf->compileCuts();

// print the cut hierarchy
cuts->print();
```

continued: TQCutFactory

```
// create an instance of the cut factory
TQCutFactory * cf = new TQCutFactory();

// define two cuts
cf->addCut(„rootCut: nLeptons == 2“);
cf->addCut(„MTCut: rootCut << MT < $MH“);

// compile the cut hierarchy
TQCompiledCut * cuts =
  cf->compileCuts(„mh = 125“);

// print the cut hierarchy
cuts->print();
```

continued: TQCutFactory

```
// create an instance of the cut factory
TQCutFactory * cf = new TQCutFactory();

// define two cuts
cf->addCut(„rootCut: nLeptons == 2 : weight“);
cf->addCut(„CutZVeto: rootCut <<
  { $LEPCH!=‘em‘ ? abs(Mll-91.1876)>15 : 1 }“);

// compile the cut hierarchy
TQCompiledCut * cutsEE =
  cf->compileCuts(„lepch = ‘ee‘“);
TQCompiledCut * cutsEM =
  cf->compileCuts(„lepch = ‘em‘“);

// print the cut hierarchy
cutsEE->print();
cutsEM->print();
```

apply an event weight for events passing this cut

samples need to be tagged with „usemcweights“ = true if event weights should be applied

Analysis jobs (`TQAnalysisJob`)

The `TQAnalysisJob` class is a representation of a simple analysis job to be associated to a certain selection cut (represented by an instance of the `TQCompiledCut` class).

There are several pre-implemented analysis job classes (inheriting from the `TQAnalysisJob` class) available:

- `TQHistoMakerAnalysisJob` ← filling histograms
- `TQCutflowAnalysisJob` ← counting events
- `TQScanAnalysisJob` ← scanning windows
- `TQEventlistAnalysisJob` ← creating event list
- `TQCopyTreeAnalysisJob` ← writing small ntuples

Example 6: Cutflow analysis job

```
// get a cut hierarchy (e.g. from example 5)
...

// create a cutflow analysis job
TQCutflowAnalysisJob * cutflowJob =
  new TQCutflowAnalysisJob();

// add the cutflow analysis job to cuts of the hiearchy
cuts->addAnalysisJob(cutflowJob, „rootCut*“);

// print the cut hierarchy
cuts->print();
```

will add the job to „rootCut“ and every descendant of „rootCut“

Example 7: Histogram analysis job

```
// get a cut hierarchy (e.g. from example 5)
...

// create a cutflow analysis job
TQHistoMakerAnalysisJob * histoJob =
  new TQHistoMakerAnalysisJob();

// define („book“) a histogram
histoJob->bookHistogram(
  „TH1D(‘lepPt0‘, ‘‘, 20, 0., 200.) << „
  „(leptPt0 : ‘leading lepton pt [GeV]‘)“);

// add the histogram maker analysis job to cuts of the hiearchy
cuts->addAnalysisJob(histoJob, „rootCut*“);

// print the cut hierarchy
cuts->print();
```

use default ROOT syntax

Example 7: Histogram analysis job

```
// get a cut hierarchy (e.g. from example 5)
...

// create a cutflow analysis job
TQHistoMakerAnalysisJob * histoJob =
  new TQHistoMakerAnalysisJob();

// define („book“) a histogram
TH1D(‘lepPt0‘, ‘‘, 20, 0., 200.) << (leptPt0 : ‘leading lepton pt [GeV]‘)
  „(leptPt0 : ‘leading lepton pt [GeV]‘)“);

// add the histogram maker analysis job to cuts of the hiearchy
cuts->addAnalysisJob(histoJob, „rootCut*“);

// print the cut hierarchy
cuts->print();
```

expression of distribution

title of x axis

`TQAnalysisSampleVisitor` class

The `TQAnalysisSampleVisitor` class is an inheritor of the `TQSampleVisitor` class designed to run the analysis jobs associated to a cut hierarchy on the sample hierarchy.

Set the base (root) cut of the cut hierarchy

- **`visitor->setBaseCut(baseCut);`**

instance of `TQCompiledCut`

The jobs are executed by visiting the sample hierarchy

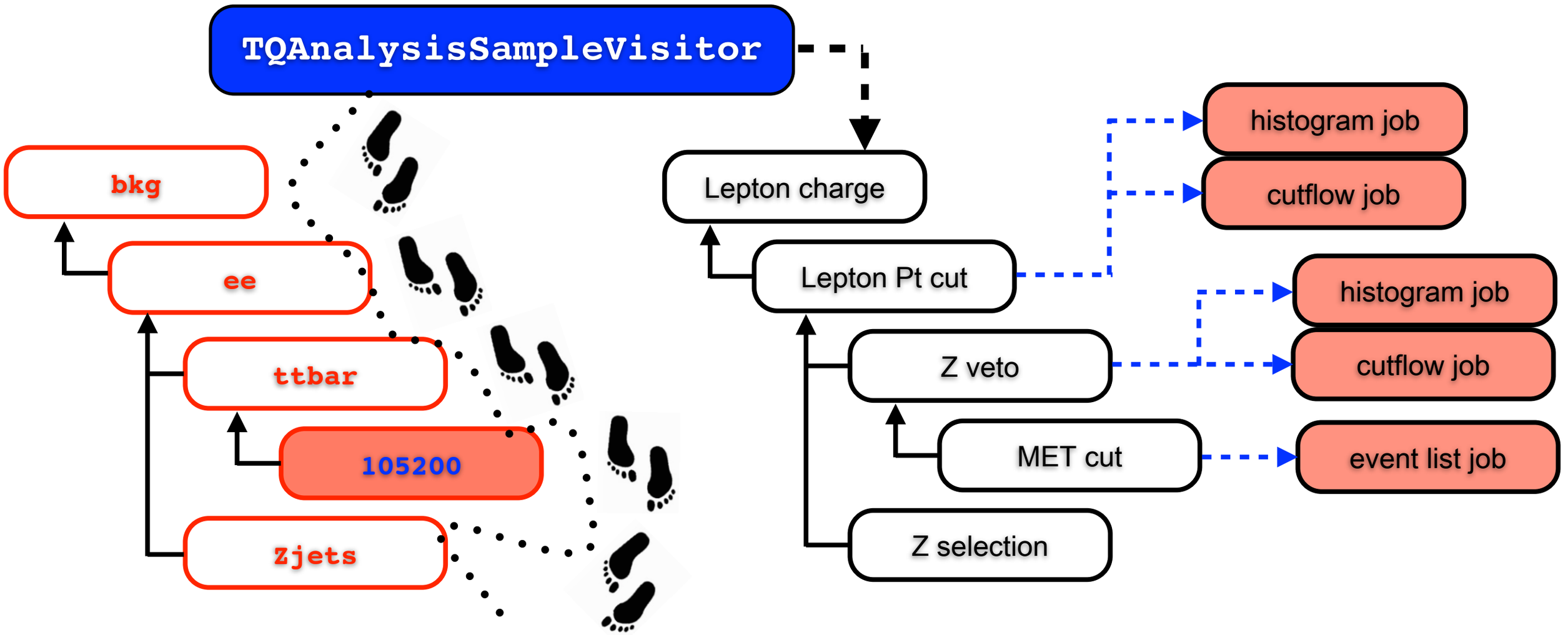
- **`samples->visitMe(visitor);`**

```
+ top
  + ttbar
    + 105200                              [   OK   ]       312'726        8.41
    + 105204                              [ failed ]            --        0.00  failed to get tree token
  + singletop
    + 117360                              [   OK   ]           274        0.02
    + 117361                              [   OK   ]           265        0.02
```

Summarizing default analysis tools

The analysis to be performed on samples in the sample folder hierarchy is defined by

- a tree of selection cuts
- analysis jobs associated to the cuts



Running a specific analysis

Many analysis requirements are met by the implementation of the `TQAnalysisSampleVisitor` and the set of analysis job classes.

However, there will be cases in which you need to implement your own sample visitor class in order to address the specifics of your analysis.

Define a new class (outside the library) implement at least the method being called when the sample visitor is visiting a sample:

- **`TMyVisitor::visitSample(...)`**

Example 8: A new sample visitor

```
// define a new class inheriting from the TQSampleVisitor class
class TMyVisitor : public TQSampleVisitor {

  public:
    // this is the most important method to reimplement
    Int_t visitSample(TQSample * sample, TString * message) {

      // access the sample‘s tree by getting a tree token
      TQToken * treeToken = sample->getTreeToken();

      if (treeToken) {
        TTree * tree = (TTree*)treeToken->getContent();

        // in this place: perform your analysis and write your
        // analysis results (to the folder hierarchy)

        // we are done: return the tree token
        sample->returnTreeToken(treeToken);
        // display the green [  OK  ]
        return visitOK;
      } else {
        // the sample didn‘t dispense a tree token => compile
        // an error message to be shown in the message column
        *message = „couldn‘t access the tree“
        // display the red [ failed ]
        return visitFAILED;
      }
    }
}
```

zoomed in: Example 8

```
// access the sample‘s tree by getting a tree token
TQToken * treeToken = sample->getTreeToken();

if (treeToken) {
  TTree * tree = (TTree*)treeToken->getContent();

  // in this place: perform your analysis and write your
  // analysis results (to the folder hierarchy)

  // we are done: return the tree token
  sample->returnTreeToken(treeToken);
  // display the green [  OK  ]
  return visitOK;
} else {
  // the sample didn‘t dispense a tree token => compile
  // an error message to be shown in the message column
  *message = „couldn‘t access the tree“
  // display the red [ failed ]
  return visitFAILED;
}
```

Exercise 12: Specific analysis

Implement a sample visitor class selecting central jets (e.g. |η| < 2.5) of every event in the sample and creating a histogram of the jet pt of those jets. Store the histogram in the sample folder hierarchy.

Some comments/hints:

- use the class skeleton „`TMySampleVisitor.cxx`“
- jet η branch: „`m_jet_eta`“, jet pt (in MeV) branch: „`m_jet_pt`“ (both `std::vector<float>`)
- for simplicity: don‘t apply any event weights or event cuts
- don‘t forget to apply the normalization factor of the sample
- use `histo->SetDirectory(0);` for your histogram
- use „`Run_TMySampleVisitor.C`“ to run your code

Some final remarks

- The `HWWAnalysisCode` library contains much more features and details than could be presented in this tutorial
- Would like to have a detailed manual and/or reference guide (due to time constraints I wasn‘t able to write such documents yet)
- Should find at least one (better two) person getting involved in the development and maintenance of the `HWWAnalysisCode` (distribute expertise)
- Many ideas on potential future improvements and new features in mind

Thanks for your attention!