Tutorial category: Expert mode

First steps in the expert mode

Version	0.1 (beta version)
Date	30/09/2013

Official MadAnalysis 5 website : https://launchpad.net/madanalysis5/

Goals of this tutorial

- Entering the expert mode
- Handling the structure of a job folder
- Coding in C++ your first analysis within the SampleAnalyzer framework
- Compiling and running your job
- Analyzing the results of your job

Part 1

Introduction

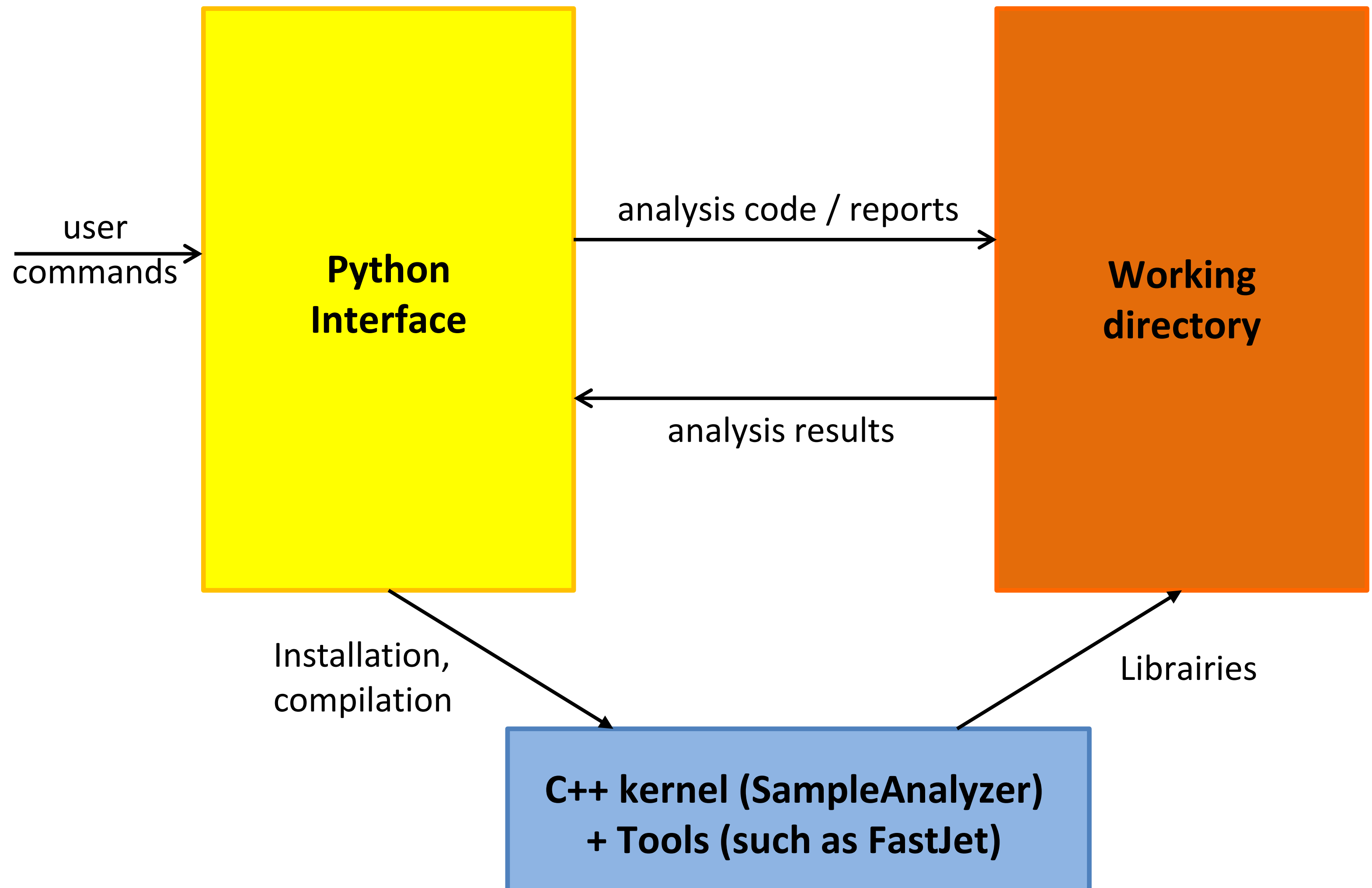
MadAnalysis 5

Requirements

- MadAnalysis 5 is installed on your system and has been launched successfully at least one time. The collection of example samples is installed too.
- Knowledge of the MadAnalysis 5 main concepts (see tutorials for beginners) and the motivations of the MadAnalysis 5 expert mode.
- Basic skills in C++ programing.
- You have chosen which text editor is your favorite ☺

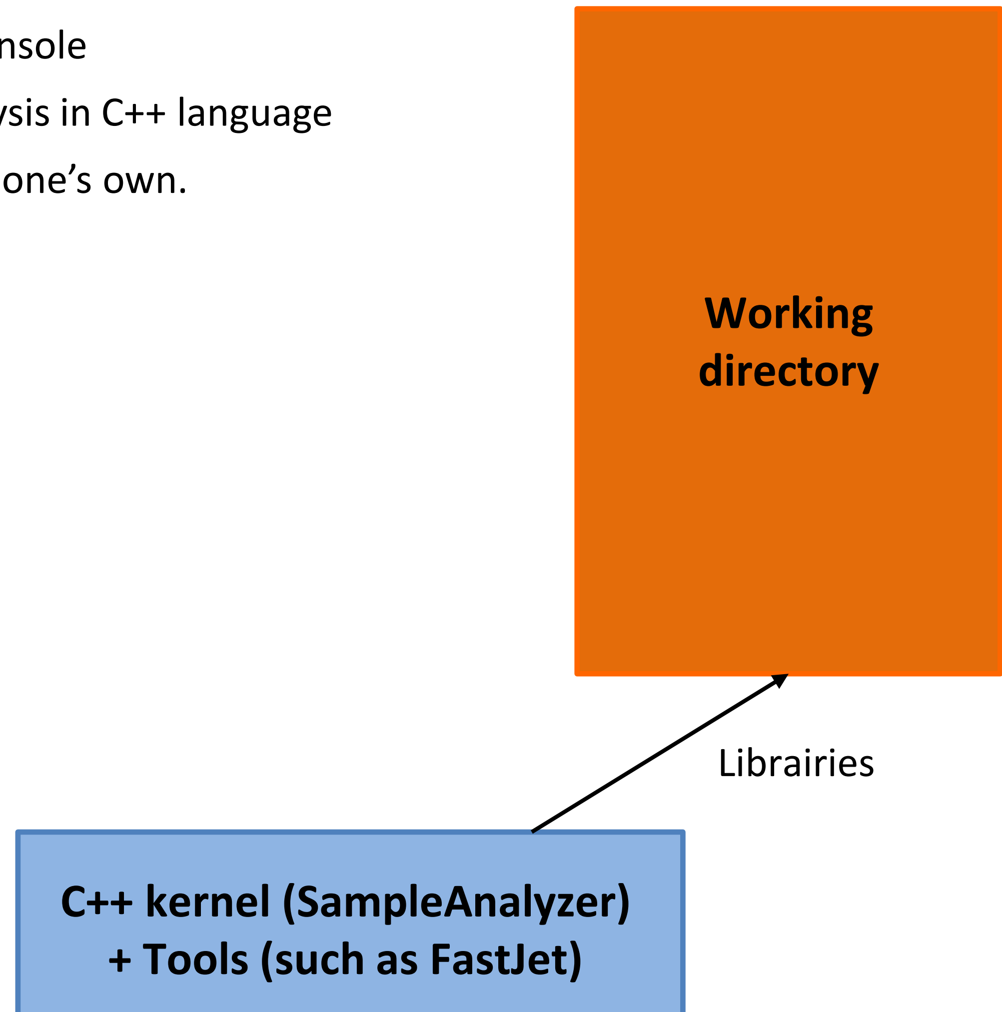
MAD Analysis 5

Reminder: the "normal" mode



What is the expert mode ?

- Not using the Python console
- Coding directly the analysis in C++ language
- Analyzing the results on one's own.

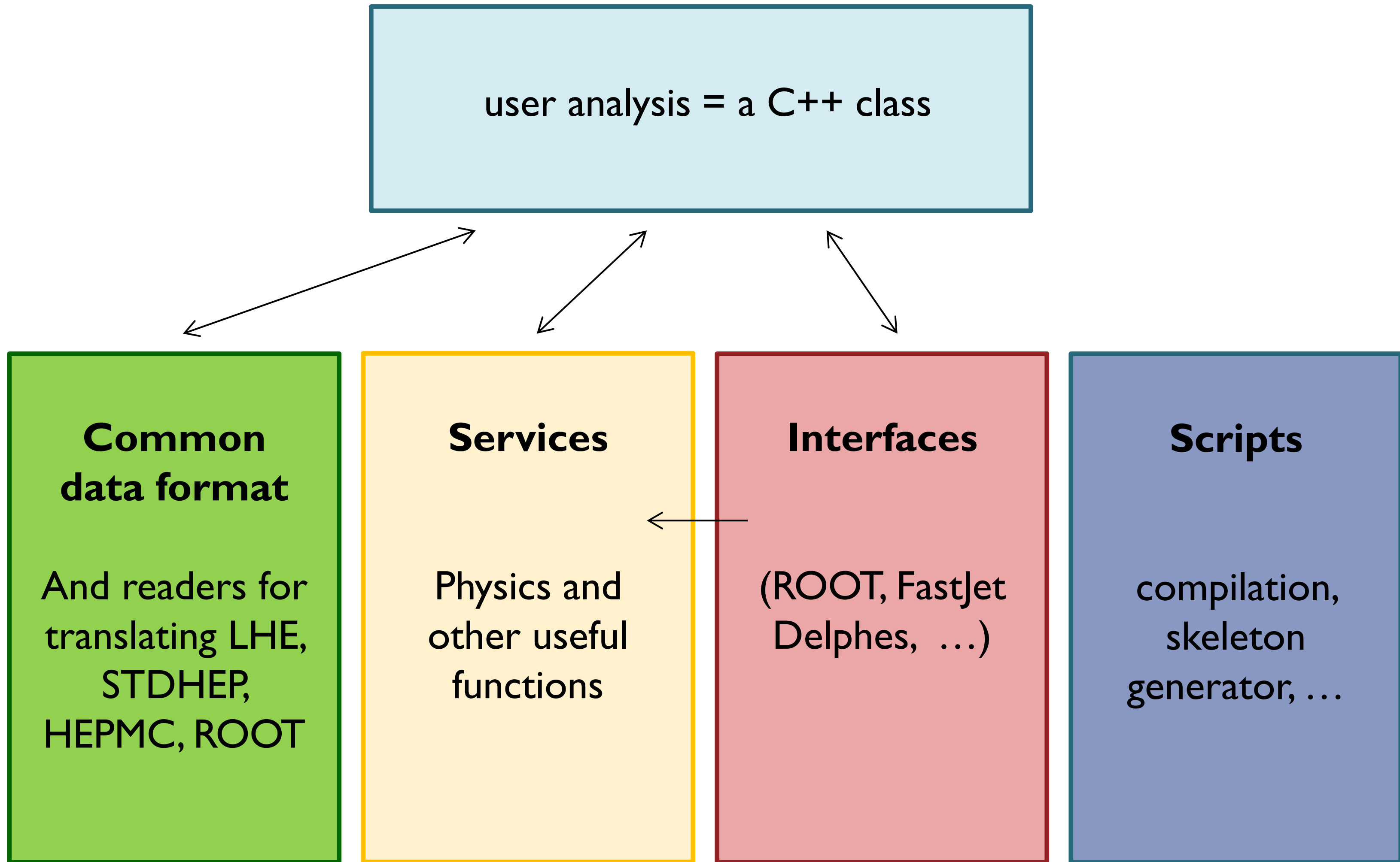


Motivations for the expert mode ?

The expert mode is motivated in several cases:

- Despite the potential of the Python console, the analysis planned is too sophisticated.
- The user would like to plug to MadAnalysis 5 an external package for which no interface is provided.
- The need to implement a specific output format for the analysis results (histograms, cuts, …) or for event data.
- The case where too many datasets must be analyzed. The user could use MadAnalysis 5 though standalone jobs and could take profit from computing resources such as a cluster or the Grid.
- Linking SampleAnalyzer library to a software for generating plots. [AVAILABLE SOON]

Expert mode = developer-friendly



Entering the expert mode

To begin an analysis in the expert mode, the user must launch MadAnalysis 5 with the argument `-e` or equivalently `--expert`.

```
./bin/ma5 -e
```

MadAnalysis 5 will initialize itself normally but at the end, the `ma5>` prompt is replaced by a series of questions. You answer will help MadAnalysis 5 to generate the proper in order to know what you would like.

First question:

```
Welcome to the expert mode of MadAnalysis
Please enter a folder for creating an empty SampleAnalyzer job
```

Just specifying the name of the working directory you would like to create.

Second question:

```
A new class called 'user' will be created.
Please enter a title for your analyzer :
```

At this step of the tutorial, this name is insignificant. Only for the display.

Entering the expert mode

Assuming you answer is ‘MyAnalysis’ to the questions 1 and 2, a working directory called `MyAnalysis` is created and contains an empty analysis called `MyAnalysis` and scripts (required in particular for compiling).

Some guidelines are given at the screen in order to survive in the expert mode. They can be considered as a reminder of the present tutorial.

```
Creating folder '/grid_mnt/home/econte/MA5/v1.1.9beta/MyAnalysis'...
  Copying required 'SampleAnalyzer' source files...
  Writing an empty analysis...
  Writing a Makefile...
```

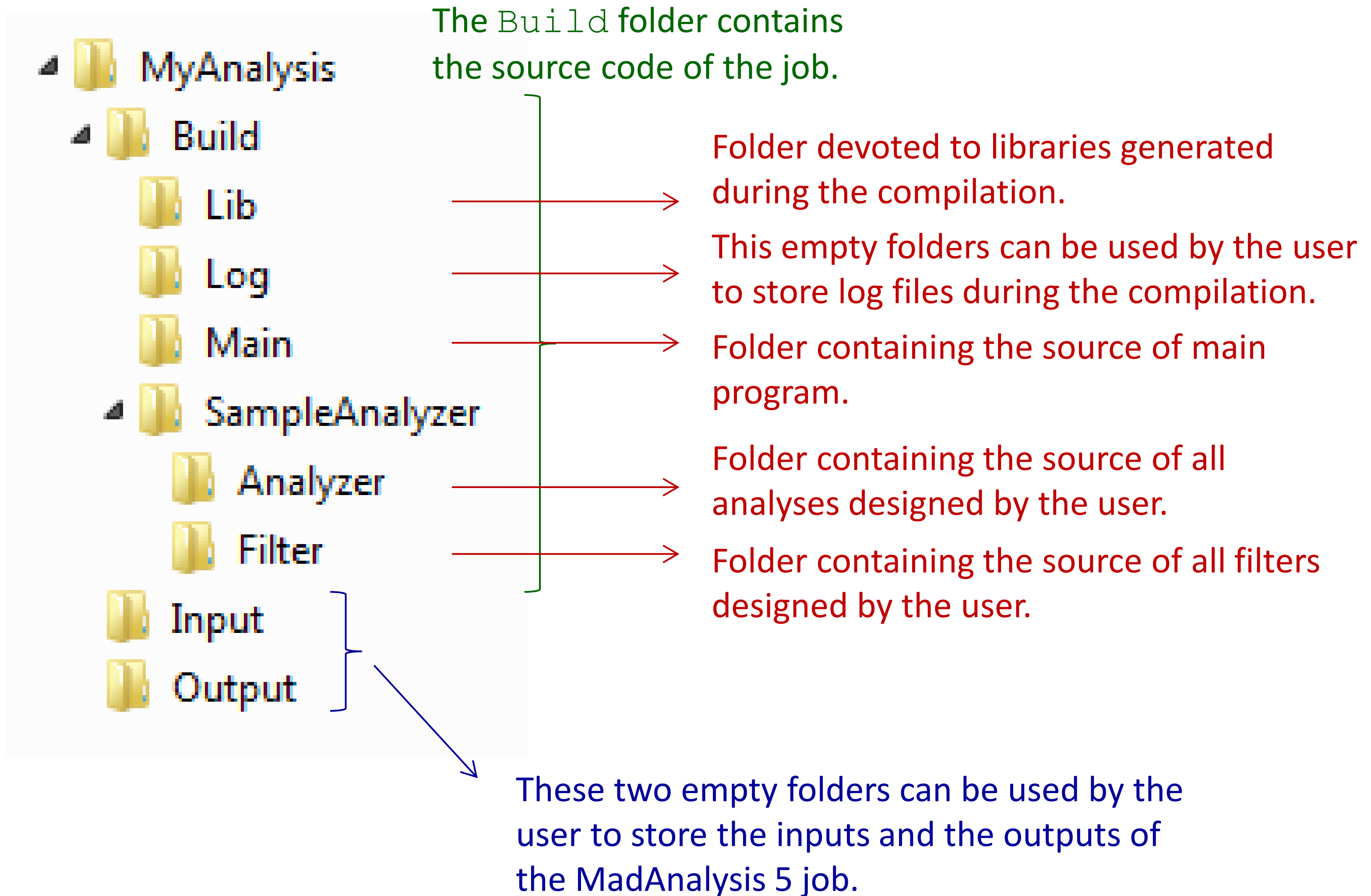
Another way to enter the expert mode:

It is possible also to modify a working directory (and its files) generated by the Python interface in the «normal» mode.

Part 2

Launching the template analysis

Structure of the working directory



Setting your environment

Before beginning to work, the environment variables required by MadAnalysis 5 must be set. To this end, entering the folder `Build` of `MyAnalysis`:

```
cd MyAnalysis/Build
```

and executing the script `setup.sh` if you use the shell BASH

```
source setup.sh
```

or the script `setup.csh` if you use the shell TCSH.

```
source setup.csh
```

If the script has been properly executed, the following message must appear:

```
--------------------------------------------------------
    Your environment is properly configured for MA5
--------------------------------------------------------
```

This first step must be carried out each time you start a new shell session.

Building your job

To build the job, you must type inside the `Build` folder

```
make
```

As all C++ programs, the building of the job is made up of two parts: compilation and linking. If the building is successful, an executable file called `MadAnalysis5Job` will be created.

When the executable is built, you can save disk space by removing the intermediate files produced during the compilation (object files). This purpose can be performed by issuing

```
make clean
```

You have also the option to remove all files produced during the building phase (the executable MadAnalysis5Job also) and to come back to the initial configuration.

```
make mrproper
```

Launching job

The executable file MadAnlysis5job built in the Build folder is fully independent from the place where it is. It can be moved in any folder of your choice.

Before launching the job, the list of samples you would like to process must be specified. It can be done by creating a text file containing the list of the files. The syntax is simple: one line by sample. Be careful the wildcard characters * and ? are not allowed.

Considering the example of a text file called input.txt containing the lines:

```
/opt/cms/data1/zz_sample1.lhe.gz
/opt/cms/data1/zz_sample2.lhe.gz
/opt/cms/data1/zz_sample3.lhe.gz
```

The job can be launched by issuing

```
./MadAnalysis5job input.txt
```

Part 3

Understanding the template analysis code

Structure of the folder Build/SampleAnalyzer/Analyzer

In this tutorial, we focus only on the analysis folder.

Source file where the different analyses are declared

The analysis template that MadAnalysis 5 has created for you

Focus on analysisList.cpp

This is the content of `analysisList.cpp` produced by MadAnalsis 5:

```
#include "SampleAnalyzer/Analyzer/user.h"
#include "SampleAnalyzer/Analyzer/AnalyzerManager.h"
#include "SampleAnalyzer/Service/LogStream.h"
using namespace MA5;
#include <stdlib.h>

// -----------------------------------------------------------------------
// BuildTable
// -----------------------------------------------------------------------
void AnalyzerManager::BuildUserTable()
{
  Add("user",new user);
}
```

Nothing to do here !

Focus on the analysis template

Structure of the header file

The class must heritate from the mother class `AnalyzerBase`

```
class user : public AnalyzerBase
{
  INIT_ANALYSIS(user,"user")

 public:

  virtual bool Initialize(const MA5::Configuration& cfg, const
                          std::map<std::string,std::string>& parameters);

  virtual void Finalize(const SampleFormat& summary,
                        const std::vector<SampleFormat>& files);

  virtual void Execute(SampleFormat& sample, const EventFormat& event);

 private:
};
```

MAD Analysis 5

Focus on the analysis template

First steps in the expert mode

Structure of the header file

```
class user : public AnalyzerBase
{
  INIT_ANALYSIS(user,"user")

 public:

  virtual bool Initialize(const MA5::Configuration& cfg, const
                                std::map<std::string,std::string>& parameters);

  virtual void Finalize(const SampleFormat& summary,
                              const std::vector<SampleFormat>& files);

  virtual void Execute(SampleFormat& sample, const EventFormat& event);

 private:
};
```

The preprocessor macro `INIT_ANALYSIS` allows to set the name of the analysis. The constructors and destructor are also encapsulated in this macro.

Focus on the analysis template

Structure of the header file

```
class user : public AnalyzerBase
{
  INIT_ANALYSIS(user,"user")

 public:

  virtual bool Initialize(const MA5::Configuration& cfg, const
                          std::map<std::string,std::string>& parameters);

  virtual void Finalize(const SampleFormat& summary,
                        const std::vector<SampleFormat>& files);

  virtual void Execute(SampleFormat& sample, const EventFormat& event);

 private:
};
```

The method `Initialize` is executed one time before beginning to read the events.

It can be used to initialize global variables, or histograms, to allocate memory …

Arguments = configuration of MadAnalysis 5

Returned value = True (successful initialization) or False (failed initialization → the job is stopped)

Focus on the analysis template

Structure of the header file

```
class user : public AnalyzerBase
{
  INIT_ANALYSIS(user,"user")

 public:

  virtual bool Initialize(const MA
                                std::map<std::string,std::string>& parameters);

  virtual void Finalize(const SampleFormat& summary,
                        const std::vector<SampleFormat>& files);

  virtual void Execute(SampleFormat& sample, const EventFormat& event);

 private:
};
```

The method `Finalize` is executed one time after finishing to read all the events.

It can be used to compute results, to write plots, free allocated memory…

Arguments = general information about each event file (*e.g.* cross section) and summarized information (e.g. mean cross section)

No returned value.

MAD Analysis 5

Focus on the analysis template

Structure of the header file

```
class user : public AnalyzerBase
{
  INIT_ANALYSIS(user,"user")

 public:

  virtual bool Initialize(const MA5::Configuration& cfg, const
                               std::map<std::string,std::string>& parameters);

  virtual void Finalize(const SampleFormat& summary,
                        const std::vector<SampleFormat>& files);

  virtual void Execute(SampleFormat& sample, const EventFormat& event);

 private:
};
```

The method `Execute` is each time after reading one event.

It can be used to fill histograms with event data or to apply selection cuts.

Arguments = general information about the current event file (*e.g.* cross section) and the content of the current event.

No returned value.

Focus on the analysis template

Structure of the source file

```
// -----------------------------------------------------------------------
// Initialize
// function called one time at the beginning of the analysis
// -----------------------------------------------------------------------
bool user::Initialize(const MA5::Configuration& cfg, const
                      std::map<std::string,std::string>& parameters)
{
  cout << "BEGIN Initialization" << endl;
  // initialize variables, histos
  cout << "END   Initialization" << endl;
  return true;
}
```

The content of the `Initialize` method can be changed in the source file.

MAD Analysis 5

Focus on the analysis template

First steps in the expert mode

Structure of the source file

```
// -----------------------------------------------------------------------------
// Finalize
// function called one time at the end of the analysis
// -----------------------------------------------------------------------------
void user::Finalize(const SampleFormat& summary, const
                    std::vector<SampleFormat>& files)
{
  cout << "BEGIN Finalization" << endl;
  // saving histos
  cout << "END   Finalization" << endl;
}
```

The content of the `Finalize` method can be changed in the source file.

Focus on the analysis template

Structure of the source file

```
// -----------------------------------------------------------------------
// Execute
// function called each time one event is read
// -----------------------------------------------------------------------
void user::Execute(SampleFormat& sample, const EventFormat& event)
{
  // ***********************************************************************
  // Example of analysis with generated particles
  // Concerned samples: LHE/STDHEP/HEPMC
  // ***********************************************************************
  /* …. */

  // ***********************************************************************
  // Example of analysis with reconstructed objects
  // Concerned samples: LHCO or STDHEP/HEPMC after fast-simulation
  // ***********************************************************************
   /* …. */
}
```

Focus on the analysis template

Structure of the source file

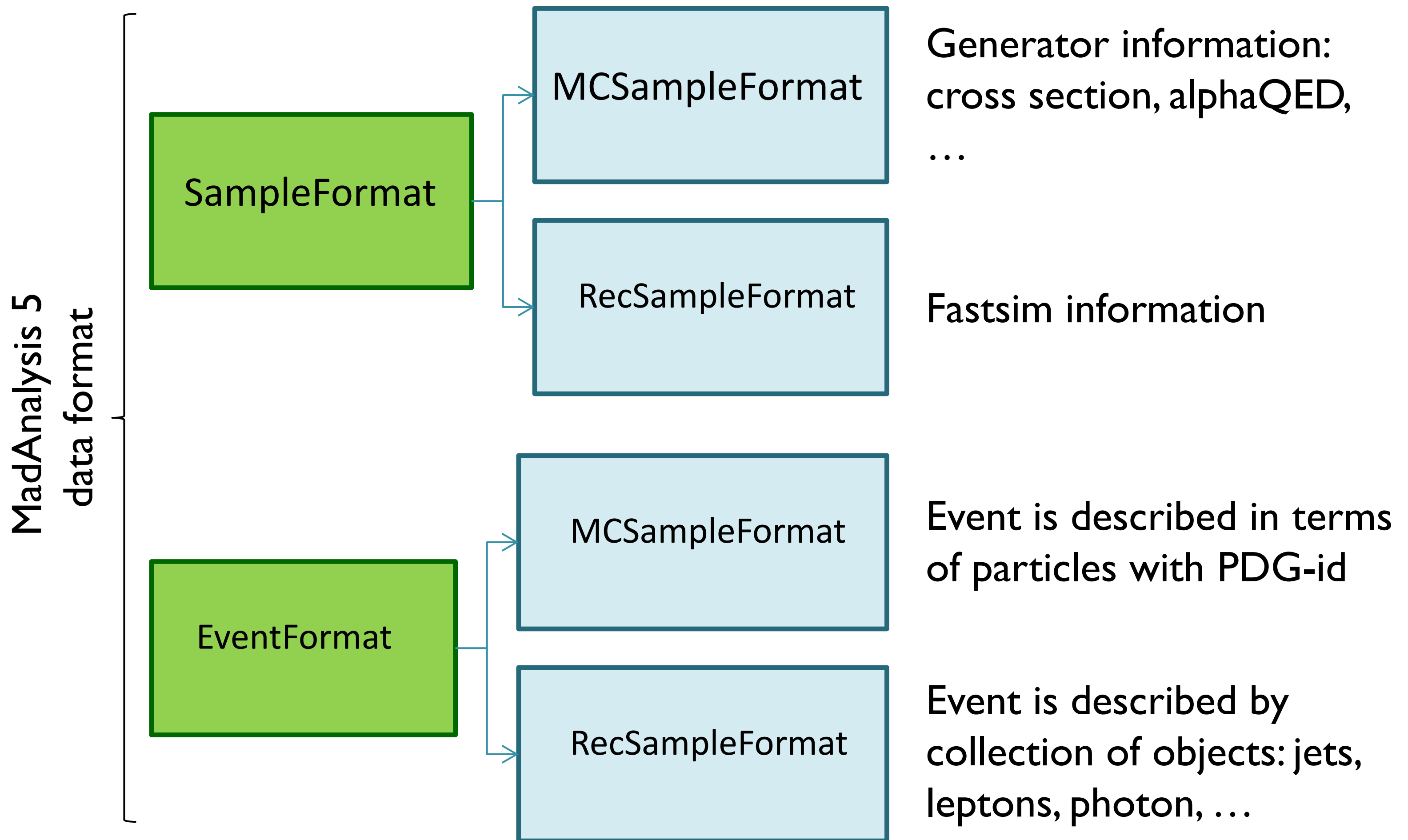
```
// ---------------------------------
// Execute
// function called each time one eve
// ---------------------------------
void user::Execute(SampleFormat& sam
{
  // ***************************************************************************
  // Example of analysis with generated particles
  // Concerned samples: LHE/STDHEP/HEPMC
  // ***************************************************************************
  /* …. */

  // ***************************************************************************
  // Example of analysis with reconstructed objects
  // Concerned samples: LHCO or STDHEP/HEPMC after fast-simulation
  // ***************************************************************************
   /* …. */
}
```

For the Execute method, 2 complete examples are given. Initially they are commented out. To use one of them, you have just to uncomment it.

The first example is related to partonic or hadronic events. The second one is designed for reconstructed events.

Few words about the data format

Before studying the two examples, it is necessary to present the data format used.



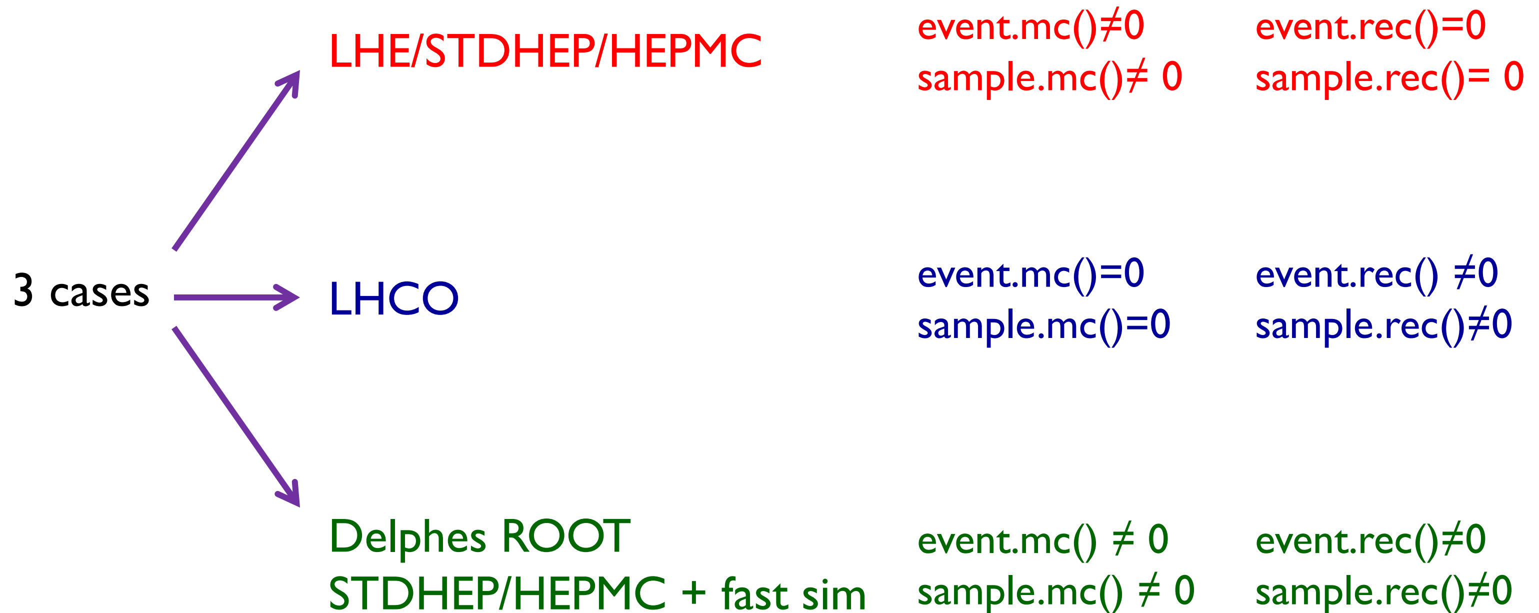
Few words about the data format

How to access the dataformat components ?

```
// -----------------------------------------------------------------------
// Execute
// function called each time one event is read
// -----------------------------------------------------------------------
void user::Execute(SampleFormat& sample, const EventFormat& event)
{
  const MCEventFormat* eventmc = event.mc();
  const RecEventFormat* eventrec = event.rec();

  const MCSampleFormat* samplemc = sample.mc();
  const RecSampleFormat* samplerec = sample.rec();

  ...
}
```

Few words about the data format

Data format versus the event sample format:



Few words about the data format

Content of MCEventFormat:

- `weight:` event-weight of the event
- `processId:` identity code of the physics process
- `alphaQED`
- `alphaQCD`
- `PDFscale:` Q
- `x.first:` x1
- `x.second:` x2
- `xpdf.first:` x1f(x1)
- `xpdf.second:` x2f(x2)
- one collection of generated `particles`
- `met` : missing transverse energy (**TLorentzVector**)
- `mht` : missing transverse hadronic energy (**TLorentzVector**)
- `tet` : total transverse energy (**double**)
- `tht` : total transverse hadronic energy (**double**)

Few words about the data format

Content of MCEventFormat: focus on `particles`

- one collection `particles`. For each particle, we have:
  - `pdgid` : PDG identity code (**integer**)
  - `statuscode:` status code (**integer**)
  - `momentum`: four-momentum (**ROOT TLorentzVector**)
  - `position`: position of the decaying vertex (**ROOT TVector**)
  - `mother`: pointer to the mother particle
  - `daughters`: collection of pointers to the daughter particles

Few words about the data format

Content of RecEventFormat:

- `jets:` collection of jets
- `muons:` collection of muons
- `electrons:` collection of electrons
- `taus:` collection of hadronically-decaying taus
- `photons:` collection of photons

- `met` : missing transverse energy (**TLorentzVector**)
- `mht` : missing transverse hadronic energy (**TLorentzVector**)
- `tet` : total transverse energy (**double**)
- `tht` : total transverse hadronic energy (**double**)

Few words about the data format

Content of RecEventFormat. Focus on the jets collection:

- **`jets:`**
  - **`momentum:`** four-momentum
  - **`ntrack:`** number of tracks (charged stable particles) in the jet
  - **`HEoverEE`**: adronic energy over electrogmagnetic energy
  - **`EEoverHE`**

  - **`btag:`** collection of muons
  - **`true_btag:`** collection of electrons
  - **`true_ctag:`** collection of hadronically-decaying taus

  - **`mc:`** pointer to the parton initiated the jet (MC particle)
  - **`constituents:`** collection of pointers to MC particles contained in the jets

Few words about the Physics service

SampleAnalyzer provides useful functions, gathered by topics into services. The first service to learn is the physics one.

The physics service is a C++ singleton refered by a pointer called: **`PHYSICS`**

Example1 : using the Physics service for decoding the status code

```
const MCParticleFormat& part = [...]
cout << "final state ?" << PHYSICS->IsFinalState(part) << endl;
cout << "intermediate state ?" << PHYSICS->IsInterState(part) << endl;
cout << "initial state ?" << PHYSICS->IsInitialState(part) << endl;
```

Example2 : moving a particle `part1` to the rest frame of particle `part2`

```
PHYSICS->ToRestFrame(part1,part2);
```

Example3 : computing alphaT observable related to the event `myevent`

```
PHYSICS->AlphaT(myevent);
```

MAD Analysis 5

First steps in the expert mode

Decoding an extract of the example 1

```
if (event.mc()!=0)
{
  for (unsigned int i=0;i<event.mc()->particles().size();i++)
  {
    const MCParticleFormat& part = event.mc()->particles()[i];
    [...]

    // pdgid
    cout << "pdg id=" << part.pdgid() << endl;
    if (PHYSICS->IsInvisible(part)) cout << " (invisible particle) ";
    else cout << " (visible particle) ";
    cout << endl;

    // display kinematics information
    cout << "px=" << part.px() << " py=" << part.py() << " pz=" << part.pz()
         << " e=" << part.e() << " m=" << part.m() << endl;
    cout << "pt=" << part.pt()  << " eta=" << part.eta()
         << " phi=" << part.phi() << endl;
}
```

Decoding an extract of the example 2

```
if (event.rec()!=0)
{
  for (unsigned int i=0;i<event.rec()->electrons().size();i++)
  {
    const RecLeptonFormat& elec = event.rec()->electrons()[i];

    cout <<  "index=" << i+1
         << " charge=" << elec.charge() << endl;
    cout <<  "px=" << elec.px() << " py=" << elec.py()
         << " pz=" << elec.pz()
         << " e="  << elec.e()
         << " m="  << elec.m() << endl;
    cout << "pt=" << elec.pt()
         << " eta=" << elec.eta()
         << " phi=" << elec.phi() << endl;

    [...]
  }
}
```

Part 4

Modifying the predefined analysis

Focus on the analysis template

Structure of the source file

```
// -----------------------------------
// Execute
// function called each time one eve
// -----------------------------------
void user::Execute(SampleFormat& sa
{
  // ****************************************************************
  // Example of analysis with generated particles
  // Concerned samples: LHE/STDHEP/HEPMC
  // ****************************************************************
  /* …. */

  // ****************************************************************
  // Example of analysis with reconstructed objects
  // Concerned samples: LHCO or STDHEP/HEPMC after fast-simulation
  // ****************************************************************
   /* …. */
}
```

For the Execute method, 2 complete examples are given. To use one of them, you have just to uncomment it.

In the following, only Example 2 is considered.

We are going to learn how to do plots and apply selection cuts on events.

Histograming

Method 1: histograming based on ROOT

1) Declaring your histogram in the header file (private block)

```
TH1F * myHisto;        // No need to include ROOT headers
```

2) Initializing your histogram in the source file (Initialize function)

```
myHisto = new TH1F("electron pt", "electron pt", 100, 0, 100);
```

3) Saving your histogram into a ROOT file, in the source file (Finalize function)

```
// Creating a ROOT file for histos
TFile* myOutput = new TFile("MyOutput.root", "RECREATE");
```

```
// Saving your histogram
myOutput->cd(); myHisto->Write();
```

Reserved to great experts: delete dynamically-allocated ROOT objects

Histograming

Method 1: histograming based on ROOT

4) Filling your histogram in the source file (Execute function) with a weight = 1.

```
for (unsigned int i=0;i<event.rec()->electrons().size();i++)
{
  const RecLeptonFormat& elec = event.rec()->electrons()[i];
  myHisto->Fill(elec->pt());
}
```

4’) Filling your histogram in the source file (Execute function) with an event-weight

```
double eventweight = 1.
if (event.mc()!=0) eventweight = event.mc()->weight();
for (unsigned int i=0;i<event.rec()->electrons().size();i++)
{
  const RecLeptonFormat& elec = event.rec()->electrons()[i];
  myHisto->Fill(elec->pt(), eventweight);
}
```

Histograming

Method 2: histograming based on the SampleAnalyzer library

1) Declaring your histogram in the header file (private block)

```
// Declaring histogram array
PlotManager plots;
```

```
Histo* myHisto;
```

2) Initializing your histogram in the source file (Initialize function)

```
myHisto = plots.Add_Histo("electron pt",100,0,100);
```

3) Saving all histograms into SAF format with only one command in the source file (Finalize function)

```
plots.Write_TextFormat(out());
plots.Finalize();
```

Histograming

Method 2: histograming based on the SampleAnalyzer library

4) Filling your histogram in the source file (Execute function) with a weight = 1.

```
for (unsigned int i=0;i<event.rec()->electrons().size();i++)
{
  const RecLeptonFormat& elec = event.rec()->electrons()[i];
  myHisto->Fill(elec->pt());
}
```

4') Filling your histogram in the source file (Execute function) with an event-weight

```
double eventweight = 1.
if (event.mc()!=0) eventweight = event.mc()->weight();
for (unsigned int i=0;i<event.rec()->electrons().size();i++)
{
  const RecLeptonFormat& elec = event.rec()->electrons()[i];
  myHisto->Fill(elec->pt(), eventweight);
}
```

Exactly the same than method 1

Histograming

Comparison: Method 1 vs Method 2

ROOT-like

- Big collection of plot kinds (1D histo, 2D histo, TGraph, …)
- Histos saved in a ROOT file
- Layout of histograms must be changed with the ROOT software
- Difficulty to free allocated memory

SampleAnalyzer-like

- Only restricted to 1D histo (equivalent to TH1F) [up to now]
- Histos saved in a XML text file
- Layout of histograms can be changed by a home made program. No dedidated program available [up to now]
- Memory allocation is totally managed by SampleAnalyzer

Efficiency of a selection cut

Method 1: computation without any library

1) Declaring your counters in the header file

```
double Ninitial;    // number of initial events
double Nfiltered; // number of events after the a cut
```

Why `double` instead `unsigned int` ? See next slide.

2) Initializing your counters in the source file (Initialize function)

```
Ninitial = Nfiltered = 0.;
```

3) Computing efficiency in the source file (Finalize function)

```
double efficiency = 0;
if (Ninitial!=0) efficiency = Nfiltered/initial;
cout << "efficiency = " << efficiency << endl;
```

Efficiency of a selection cut

Method 1: computation without any library

4) Incrementing counters in the source file (Execute function) with a weight = 1.

```
Ninitial+=1;
if (event.rec()->MET().pt()>50) Nfiltered+=1;
```

4') Incrementing counters in the source file (Execute function) with an event-weight

```
double eventweight = 1.
if (event.mc()!=0) eventweight = event.mc()->weight();
Ninitial+=eventweight;
if (event.rec()->MET().pt()>50) Nfiltered+=eventweight;
```

Efficiency of a selection cut

Method 2: computation based on the SampleAnalyzer library

1) Declaring your counters in the header file (private block)

```
// Declaring counter array
CounterManager cuts;
```

2) Initializing your counters in the source file (Initialize function)

```
cuts.Initialize(1); // setting number of cuts
```

3) Saving all counters into SAF format with only one command
in the source file (Finalize function)

```
cuts.Write_TextFormat(out());
cuts.Finalize();
```

Efficiency of a selection cut

Method 2: computation based on the SampleAnalyzer library

4) Incrementing counters in the source file (Execute function) with a weight = 1.

```
Cuts.Set+=1;
if (event.rec()->MET().pt()>50) cuts[0]+=1;
```

4') Incrementing counters in the source file (Execute function) with an event-weight

```
double eventweight = 1.
if (event.mc()!=0) eventweight = event.mc()->weight();
Ninitial+=eventweight;
if (event.rec()->MET().pt()>50) Nfiltered+=eventweight;
```

Efficiency of a selection cut

Comparison: Method 1 vs Method 2

without SampleAnalyzer

- Very simple
- Management of the statistical uncertainty on efficiencies must be done manually

with SampleAnalyzer

- Manager class very similar to the one used for histogramming
- All the ingredients required for the statistical uncertainty computation are stored automatically.
  Final calculation must be done by the user [up to now].
- All the results are saved in the same SAF file than for histos.

End

About this document

- The present document is a part of the tutorial collection of the package MadAnalysis 5 (MA5 in abbreviated form). It has to be conceived to explain in a practical and graphical way the functionalities and the various options available in the last public release of MA5.

- The up-to-date version of this document, also the complete collection of tutorials, can be found on the MadAnalysis 5 website :

  https://launchpad.net/madanalysis5/

- Your feedback interests ourselves (bug reports, questions, comments, suggestions). You can contact the MadAnalysis 5 team by the email address : ma5team@iphc.cnrs.fr

Change log

Version	Date	Update
0.1	30/09/2013	Beta