Tutorial category: Expert mode

Understanding the predefined examples

Version 1.0

Date 29/10/2013

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

Goals of this tutorial

- Understanding the structure of an analysis class
- Overview of the common data format used by MadAnalysis
- Decoding the predefined examples
- Tuning the examples

Requirements

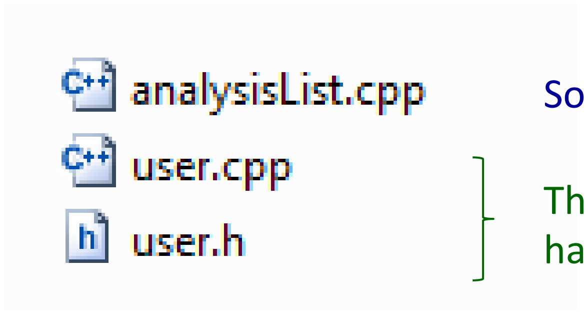
- MadAnalysis 5 is installed on your system and has been launched successfully at least one time. The collection of example samples is installed too.
- Knowledge of the MadAnalysis 5 main concepts (see tutorials for beginners).
- Basic skills in C++ programing are required here.
- You have chosen which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

Part 1

Structure of the an analysis class

Structure of the folder Build/SampleAnalyzer/Analyzer

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



analysisList.cpp

user.cpp

user.h

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:
};
```

Focus on the analysis template

Structure of the header file

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

```
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:
};
```

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.

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.

Focus on the analysis template

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

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.

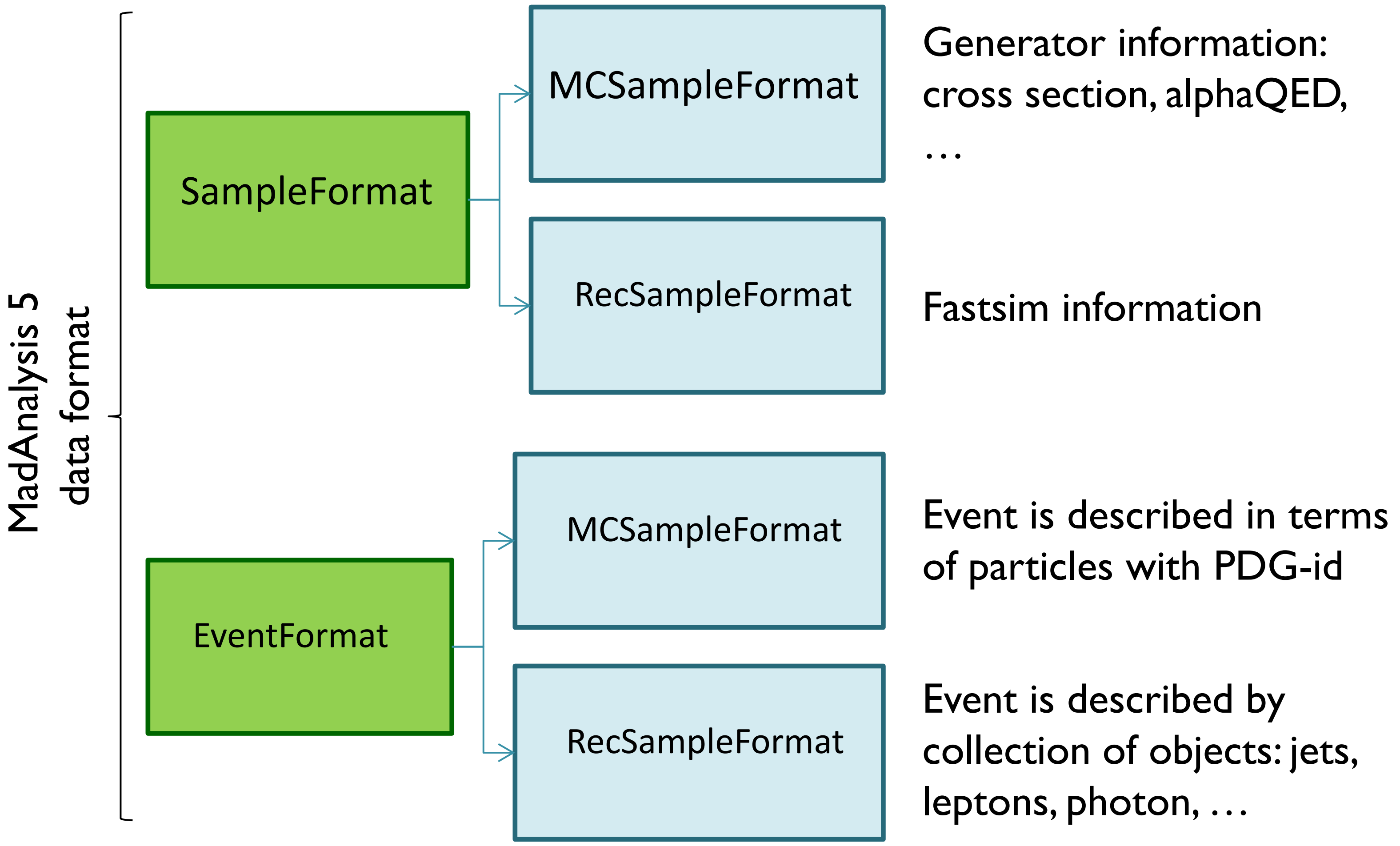
```
// --------------------------------
// Execute
// function called each time one eve
// --------------------------------
void user::Execute(SampleFormat& sam
{
  // **********************************************************************
  // Example of analysis with generated particles
  // Concerned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

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

Part 2

Few words about the data format

Few words about the data format

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


MadAnalysis 5 data format
SampleFormat
MCSampleFormat
RecSampleFormat
EventFormat
MCSampleFormat
RecSampleFormat
Generator information: cross section, alphaQED, …
Fastsim information
Event is described in terms of particles with PDG-id
Event is described by collection of objects: jets, leptons, photon, …

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:

$$\begin{tikzcd}
 & \text{LHE/STDHEP/HEPMC} \\
\text{3 cases} \arrow[ur] \arrow[r] \arrow[dr] & \text{LHCO} \\
 & \text{Delphes ROOT, STDHEP/HEPMC + fast sim}
\end{tikzcd}$$

LHE/STDHEP/HEPMC	event.mc()≠0 sample.mc()≠ 0	event.rec()=0 sample.rec()= 0
LHCO	event.mc()=0 sample.mc()=0	event.rec() ≠0 sample.rec()≠0
Delphes ROOT STDHEP/HEPMC + fast sim	event.mc() ≠ 0 sample.mc() ≠ 0	event.rec()≠0 sample.rec()≠0

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);
```

Part 3

Decoding some pieces of code

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 ev
// -------------------------------
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.

GREAT
JOB!

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://madanalysis.irmp.ucl.ac.be/wiki/tutorials

- 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 for MadAnalysis and Nsusy workshop @ Grenoble
1.0	29/10/2013	First public release