Analysis framework

Prottay Das
Niser, Jatni
Email: prottay.das@niser.ac.in

ALICE-INDIA School
IOP, Bhubhaneswar
11/11/2022

Outline

What are we gonna do today?

- ✔ Basic goal: Showing you how to write and run a minimal task to do the analysis
- ✔ No physics here

Aim?

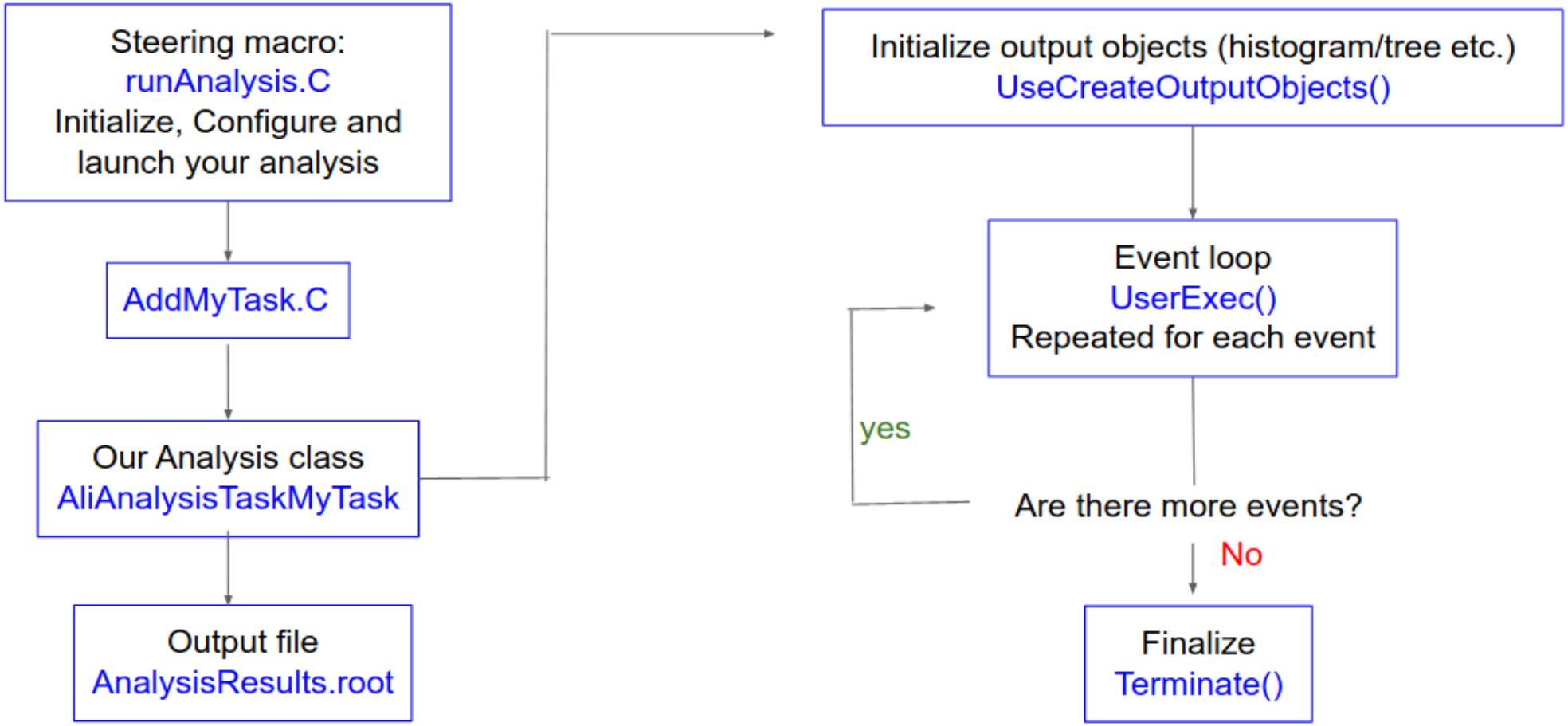
- ✔ At the end of the session, you will understand what happens in your task

Classes

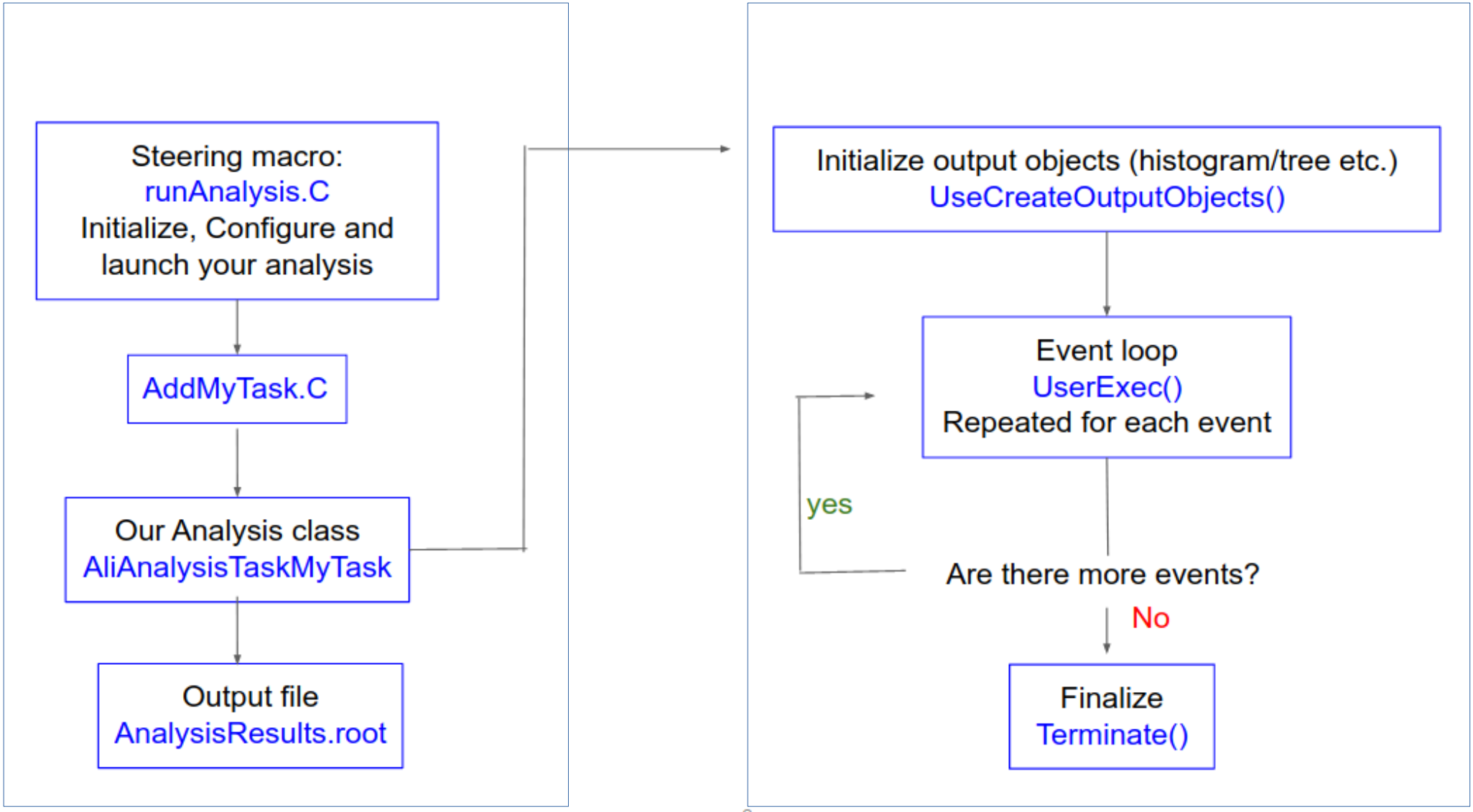
- ✔ Code in AliPhysics has the form of classes
- ✔ Our analysis task will be a c++ class
- ✔ A class contains both variables and functions called members
- ✔ Classes are nice as they can be derived from one-another

```
class Rectangle
{
int width , height ;
int GetArea ()
{
return (height * width) ;
}
}
```

ALICE analysis workflow



ALICE analysis workflow



Analysis manager

Physics analysis

Template for your class

- ✔ All analysis tasks are derived from AliAnalysisTaskSE class

- ✔ Provides us with a common framework (same base method) to do our analysis

```
AliAnalysisTaskSE::AliAnalysisTaskSE();//constructor
AliAnalysisTaskSE::AliAnalysisTaskSE(const char*);//constructor
AliAnalysisTaskSE::~AliAnalysisTaskSE();//destructor
AliAnalysisTaskSE::UserCreateOutputObjects();//histograms/TTree/TProfile
AliAnalysisTaskSE::UserExec(Option_t*);//analysis
AliAnalysisTaskSE :: Terminate ( Option_t *);//finish
```

Format for writing our class

- ✔ The header file (.h) containing function prototypes
- ✔ The implementation file (.cxx) in which the code is implemented
- ✔ The AddTask.C macro which configures the analysis

Header: AliAnalysisMyTask.h

```
# ifndef AliAnalysisTaskMyTask_ H
# define  AliAnalysisTaskMyTask _ H
.
.
.
# endif
```

- ✔ These 3 lines form an include guard (sometimes called macro guard or header guard). This construction is used to avoid the problem of double inclusion of any macros

Header: AliAnalysisMyTask.h

```
# ifndef AliAnalysisTaskMyTask_ H
# define  AliAnalysisTaskMyTask _ H
# include  “AliAnalysisTaskSE.h”
class AliAnalysisTaskMyTask : public AliAnalysisTaskSE
.
.
.
# endif
```

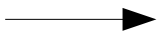
→ Class derived from AliAnalysisTaskSE

✔ These 3 lines form an include guard (sometimes called macro guard or header guard). This construction is used to avoid the problem of double inclusion of any macros

Header: AliAnalysisMyTask.h

```
# ifndef AliAnalysisTaskMyTask_ H
# define  AliAnalysisTaskMyTask _ H
# include  “AliAnalysisTaskSE.h”
class AliAnalysisTaskMyTask : public AliAnalysisTaskSE
{
public :
// two class constructors
AliAnalysisTaskMyTask();
AliAnalysisTaskMyTask( const char * name ) ;

// class destructor
virtual ~AliAnalysisTaskMyTask() ;
}
# endif
```



Class derived from AliAnalysisTaskSE

- ✔ Class constructors and destructors are special functions that are called when a new object of the class is created and destroyed respectively

Header: AliAnalysisMyTask.h

```
# ifndef AliAnalysisTaskMyTask_ H
# define  AliAnalysisTaskMyTask _ H
# include  “AliAnalysisTaskSE.h”
class AliAnalysisTaskMyTask : public AliAnalysisTaskSE
{
public :
// two class constructors
AliAnalysisTaskMyTask();
AliAnalysisTaskMyTask( const char * name ) ;

// class destructor
virtual ~AliAnalysisTaskMyTask() ;

// called once at beginning or runtime
virtual void UserCreateOutputObjects () ;
// called for each event
virtual void UserExec ( Option_t * option ) ;
// called at end of analysis
virtual void Terminate ( Option_t * option ) ;
}
# endif
```

Class derived from AliAnalysisTaskSE

These functions form the heart of our analysis

Header: AliAnalysisMyTask.h

```
# ifndef AliAnalysisTaskMyTask_ H
# define  AliAnalysisTaskMyTask _ H
# include  “AliAnalysisTaskSE.h”
class AliAnalysisTaskMyTask : public AliAnalysisTaskSE
{
public :
// two class constructors
AliAnalysisTaskMyTask();
AliAnalysisTaskMyTask( const char * name ) ;

// class destructor
virtual ~AliAnalysisTaskMyTask() ;

// called once at beginning or runtime
virtual void UserCreateOutputObjects () ;
// called for each event
virtual void UserExec ( Option_t * option ) ;
// called at end of analysis
virtual void Terminate ( Option_t * option ) ;

ClassDef ( AliAnalysisTaskMyTask , 1) ;
}
# endif
```

→ Class derived from AliAnalysisTaskSE

- ClassDef is a C macro that must be used if your class derives from TObject
- The version number 0 disables I/O for the class, so start from 1 always

Header: Adding histogram

```
class AliAnalysisTaskMyTask : public AliAnalysisTaskSE {

 public:

  //-----> Mandatory Functions:
  AliAnalysisTaskMyTask();
  AliAnalysisTaskMyTask(const char *name);
  virtual ~AliAnalysisTaskMyTask();
  virtual void UserCreateOutputObjects();
  virtual void UserExec(Option_t * /*option*/);


 protected:

 private:

  AliVEvent             *fVevent;            //! event
  AliESDEvent           *fESD;               //! esd Event
  AliAODEvent           *fAOD;               //! aod Event
  AliPIDResponse        *fPIDResponse;       //! PID Handler
  AliMultSelection      *fMultSelection;     //! For Centrality
  TList                 *fListHist;          //! OutputList


  TH1F                  *fHistpt;    //! pt histogram
```

- ✔ Pointers to objects that are initialized at runtime (in UserCreateOutputObjects) should have //! (transient member not steamed) at the end

Implementation: AliAnalysisTaskMyTask.cxx

```
#include "AliAnalysisTaskMyTask.h" //your analysis task
using std::cout;   //std namespace for things like cout
using std::endl;
ClassImp(AliAnalysisTaskMyTask) //necessary for root


//.............default constructor, don't allocate memory here
//this is used by root for IO purposes..................
AliAnalysisTaskMyTask::AliAnalysisTaskMyTask():
fVevent(NULL), fESD(NULL), fAOD(NULL), fPIDResponse(NULL), fMultSelection(NULL),
fListHist(NULL),fHistpt(NULL)
{
}

//..................constructor.....................................
AliAnalysisTaskMyTask::AliAnalysisTaskMyTask(const char *name): AliAnalysisTaskSE(name),
fVevent(NULL), fESD(NULL), fAOD(NULL), fPIDResponse(NULL), fMultSelection(NULL),fListHist(NULL),fHistpt(NULL)
{
  //Must be here:
  DefineInput(0,TChain::Class());//define the input of the analysis:in this case we take a chain of events created by analysis manager autom
atically
  DefineOutput(1,TList::Class());//define the output of the analysis:you can add more output objects by calling DefineOutput(2,classname::Cl
ass())

}

//_____________________ destructor _____________________
AliAnalysisTaskMyTask::~AliAnalysisTaskMyTask()
{
  if(fListHist)      delete fListHist;
}
```

- ✔ In constructors, we initialize members to their default values and tell the task what the input and output is

Implementation: AliAnalysisTaskMyTask.cxx

- ✔ Initialization of output objects

```
//_____________ Define Histograms _____________
void AliAnalysisTaskMyTask::UserCreateOutputObjects()
{

  //Get The Analysis Manager and Input Handler:
  AliAnalysisManager *mgr = AliAnalysisManager::GetAnalysisManager();
  AliInputEventHandler *inputHandler=dynamic_cast<AliInputEventHandler*>(mgr->GetInputEventHandler());
  if (!inputHandler) {  printf("\n***** ERROR *****\n Input handler missing, Status:QUIT!\n");    exit(1);}

  //// obtain the PID response object if needed:
  fPIDResponse=inputHandler->GetPIDResponse();
  if (!fPIDResponse) {  printf("\n***** ERROR *****\n fPIDResponse missing, Status:QUIT!\n");    exit(1);}

  //creating a TList
  fListHist = new TList();   //list containing the histograms
  fListHist->SetOwner(kTRUE); //memory stuff:

  /// creating a pt histogram and adding it to TList:
  fHistpt = new TH1F("fHistpt","fHistpt ",300,0,30);
  fListHist->Add(fHistpt);

  //adding the TList to our output file
  PostData(1,fListHist); //postdata will notify the analysis manager of changes/updates to the fOutputList object


}
```

Implementation: AliAnalysisTaskMyTask.cxx

- ✔ Event loop

```
//_____________________________ Call Event by Event ____________________________________
void AliAnalysisTaskMyTask::UserExec(Option_t*) {

  //the manager will take care of reading the events from file and with this InputEvent() function you have access to the current event

  fAOD = dynamic_cast <AliAODEvent*> (InputEvent());
  fESD = dynamic_cast <AliESDEvent*> (InputEvent()); //get an event (ESD) from the input file
  if(!(fESD || fAOD)) {
    printf("ERROR: fESD & fAOD not available\n");
    return;
  }

  //getting no. of tracks in an event
  Int_t ntracks = fESD->GetNumberOfTracks();
  if(ntracks < 4) return;                          // Minimum 4 tracks per event.

  //defining variables
  Float_t trkPt=0;
  ////////-----> Starting 1st track Loop -----------
  for(Int_t iTrack = 0; iTrack < ntracks; iTrack++) {
    //getting track
    AliVTrack   *track = (AliVTrack*)fESD->GetTrack(iTrack);
    if(!track)      continue;
    trkPt    = track->Pt();
    //fill our pt histogram
    fHistpt->Fill(trkPt);
  }///------>Track loop Ends here.<--------
  PostData(1,fListHist); //stream the results of the analysis of this event to the output manager which will take care of writing to a file
}//---------------- UserExec ----------------------
```

Add task macro

```
AliAnalysisTaskMyTask* AddTaskMyTask(const char *suffix = "")
{
  //get the manager
  AliAnalysisManager *mgr = AliAnalysisManager::GetAnalysisManager();
  //by defaut a file is open for writing, here we get the filename
  TString     outfileName = AliAnalysisManager::GetCommonFileName();

  TString list1OutName = outfileName;        // common outfile filename
  list1OutName        += ":Results";         // creating subfolder containing resultant histograms

  //create an instance of your class
  AliAnalysisTaskMyTask *taskMyTask = new AliAnalysisTaskMyTask("MyTask");
  taskMyTask->SelectCollisionCandidates(AliVEvent::kINT7); //trigger for analysis
  mgr->AddTask(taskMyTask);                        // connect the task to the analysis manager

  AliAnalysisDataContainer  *cinput = mgr->GetCommonInputContainer();  //container
  mgr->ConnectInput(taskMyTask, 0, cinput);        // connect the manager to your task

  AliAnalysisDataContainer  *cOutPut1;
  TString                   sMyOutName;
  sMyOutName.Form("SimpleTask_%s",suffix);

  cOutPut1 = (AliAnalysisDataContainer *) mgr->CreateContainer(sMyOutName,TList::Class(),AliAnalysisManager::kOutputContainer,list1OutName.D
ata());
  mgr->ConnectOutput(taskMyTask, 1, cOutPut1); //same for the output

  return taskMyTask;

}//Task Ends
```

Steering macro skeleton

```
void runAnalysis()
{

  //since we will compile a class, tell root where to look for headers
    gInterpreter->ProcessLine(".include $ROOTSYS/include");
    gInterpreter->ProcessLine(".include $ALICE_ROOT/include");

    // create the analysis manager
    AliAnalysisManager *mgr = new AliAnalysisManager("AnalysisTaskExample");
    AliESDInputHandler *esdH = new AliESDInputHandler();
    mgr->SetInputEventHandler(esdH);


    gInterpreter->ProcessLine(".x AliAnalysisTaskMyTask.cxx++g");


    AliAnalysisTaskMyTask *task = reinterpret_cast<AliAnalysisTaskMyTask*>(gInterpreter->ExecuteMacro("AddMyTask.C"));

    if(local) {
        // if you want to run locally, we need to define some input
      TChain* chain = CreateESDChain("localruntest.txt", 1);
      mgr->StartAnalysis("local",chain);
    }
```

Steering macro skeleton

```
// if we want to run on grid, we create and configure the plugin
AliAnalysisAlien *alienHandler = new AliAnalysisAlien();
// also specify the include (header) paths on grid
alienHandler->AddIncludePath("-I. -I$ROOTSYS/include -I$ALICE_ROOT -I$ALICE_ROOT/include -I$ALICE_PHYSICS/include");
// make sure your source files get copied to grid
alienHandler->SetAdditionalLibs("AliAnalysisTaskMyTask.cxx AliAnalysisTaskMyTask.h");
alienHandler->SetAnalysisSource("AliAnalysisTaskMyTask.cxx");
// select the aliphysics version. all other packages
// are LOADED AUTOMATICALLY!
alienHandler->SetAliPhysicsVersion("vAN-20181028_ROOT6-1");
// set the Alien API version
alienHandler->SetAPIVersion("V1.1x");
// select the input data
alienHandler->SetGridDataDir("/alice/data/2015/LHC15o");
alienHandler->SetDataPattern("*pass1/*AliESDs.root");
// MC has no prefix, data has prefix 000
alienHandler->SetRunPrefix("000");
// runnumber
alienHandler->AddRunNumber(246994);
// number of files per subjob
alienHandler->SetSplitMaxInputFileNumber(40);
alienHandler->SetExecutable("myTask.sh");
// specify how many seconds your job may take
alienHandler->SetTTL(10000);
alienHandler->SetJDLName("myTask.jdl");
```

Steering macro skeleton

```
    alienHandler->SetOutputToRunNo(kTRUE);
    alienHandler->SetKeepLogs(kTRUE);
    // merging: run with kTRUE to merge on grid
    // after re-running the jobs in SetRunMode("terminate")
    // (see below) mode, set SetMergeViaJDL(kFALSE)
    // to collect final results
    alienHandler->SetMaxMergeStages(1);
    alienHandler->SetMergeViaJDL(kTRUE);

    // define the output folders
    alienHandler->SetGridWorkingDir("myWorkingDir");
    alienHandler->SetGridOutputDir("myOutputDir");

    // connect the alien plugin to the manager
    mgr->SetGridHandler(alienHandler);
    if(gridTest) {
        // speficy on how many files you want to run
        alienHandler->SetNtestFiles(1);
        // and launch the analysis
        alienHandler->SetRunMode("test");
        mgr->StartAnalysis("grid");
    } else {
        // else launch the full grid analysis
        alienHandler->SetRunMode("full");
        mgr->StartAnalysis("grid");
    }
}
```