HLT tutorial: Making an HLT component

Matthias Richter

Dep. of Physics, University of Oslo

Feb 09 2012

Overview

The header file

```
#include "AliHLTProcessor.h"

class AliHLTTutorialComponent : public AliHLTProcessor {
public:
  AliHLTTutorialComponent();
  virtual ~AliHLTTutorialComponent();

  // AliHLTComponent interface functions
  const char* GetComponentID();
  void GetInputDataTypes( vector<AliHLTComponentDataType>& list);
  AliHLTComponentDataType GetOutputDataType();
  void GetOutputDataSize( unsigned long& constBase, double& inputMultiplier );
  void GetOCDBObjectDescription( TMap* const targetMap);
  AliHLTComponent* Spawn();

 protected:
  // AliHLTComponent interface functions
  int DoInit( int argc, const char** argv );
  int DoDeinit();
  int DoEvent( const AliHLTComponentEventData& evtData, AliHLTComponentTriggerData& trigData);
  int ScanConfigurationArgument(int argc, const char** argv);
  int Reconfigure(const char* cdbEntry, const char* chainId);
  int ReadPreprocessorValues(const char* modules);

  using AliHLTProcessor::DoEvent;

private:
  /** copy constructor prohibited */
  AliHLTTutorialComponent(const AliHLTTutorialComponent&);
  /** assignment operator prohibited */
  AliHLTTutorialComponent& operator=(const AliHLTTutorialComponent&);

  ClassDef(AliHLTTutorialComponent, 0)
};
```

Property methods

Property methods are used to announce the features of the component and hook it up to the HLT framework

- component id
- input and output data type
- estimated output data size
- creation of component
- required OCDB objects

Component Property methods II

```
const char* AliHLTTutorialComponent::GetComponentID()
{
  /// inherited from AliHLTComponent: id of the component
  return "TutorialComponent";
}

void AliHLTTutorialComponent::GetInputDataTypes( AliHLTComponentDataTypeList& tgtList)
{
  /// inherited from AliHLTComponent: list of data types in the vector reference
  tgtList.clear();
  tgtList.push_back(kAliHLTDataTypeESDObject);
}

AliHLTComponentDataType AliHLTTutorialComponent::GetOutputDataType()
{
  /// inherited from AliHLTComponent: output data type of the component.
  return kAliHLTDataTypeTObject;
}

void AliHLTTutorialComponent::GetOutputDataSize( unsigned long& constBase, double& inputMultiplier )
{
  /// inherited from AliHLTComponent: output data size estimator
  /// estimated size = inputMultiplier*input + constBase
  constBase=0;
  inputMultiplier=1.;
}
```

Component Property methods III

```
AliHLTComponent* AliHLTTutorialComponent::Spawn()
{
  /// inherited from AliHLTComponent: spawn function.
  return new AliHLTTutorialComponent;
}

void AliHLTTutorialComponent::GetOCDBObjectDescription(TMap* const targetMap)
{
  /// Get a list of OCDB object needed for the particular component
  if (!targetMap) return;

  targetMap->Add(new TObjString("GRP/GRP/Data"), new TObjString("GRP entry"));
}
```

Initialization DoInit

Parse array of string arguments, init and setup component for data processing

```
int AliHLTTutorialComponent::DoInit( int argc, const char** argv )
{
  // component initialization
  int iResult=0;

  // init stage 1: default values for all data members
  // TODO: insert member initializtion here

  // init stage 2: read configuration object
  // ScanConfigurationArgument() needs to be implemented
  // TODO: choose correct OCDB object path
  TString cdbPath="HLT/ConfigSample/"; cdbPath+=GetComponentID();
  iResult=ConfigureFromCDBTObjString(cdbPath);

  // init stage 3: read the component arguments
  if (iResult>=0) {
    iResult=ConfigureFromArgumentString(argc, argv);
  }

  if (iResult>=0) {
    // implement the component initialization
  }

  if (iResult<0) {
    // implement cleanup
  }

  return iResult;
}
```

Parsing of arguments ScanConfigurationArgument

The AliHLTComponent base class implements a parsing loop for argument strings and arrays of strings, invoked by ConfigureFromArgumentString/ConfigureFromCDBTObjString.

Overloaded function `ScanConfigurationArgument` decodes the arguments.

```
int AliHLTTutorialComponent::ScanConfigurationArgument(int argc, const char** argv)
{
  // Scan configuration arguments
  // Return the number of processed arguments
  //        -EPROTO if argument format error (e.g. number expected but not found)
  //

  int i=0;
  TString argument=argv[i];

  if (argument.IsNull()) return 0;

  // -argument1 parameter: keyword plus 1 parameter
  if (argument.CompareTo("-argument1")==0) {
    if (++i>=argc) return -EINVAL;
    HLTInfo("got \'-argument1\' argument: %s", argv[i]);
    return 2; // keyword + 1 argument
  }

  // -argument2: one keyword
  if (argument.CompareTo("-argument2")==0) {
    HLTInfo("got \'-argument2\' argument");
    return 1; // only keyword
  }

  return 0;
}
```

Event Processing DoEvent

```
int AliHLTTutorialComponent::DoEvent(const AliHLTComponentEventData& /*evtData*/,
      AliHLTComponentTriggerData& /*trigData*/)
{
  // event processing function
  if (!IsDataEvent()) return 0; // skip if it is not a normal data event

  TObjArray output; // an array for publishing the tracks as separate output
  AliInfoClass(Form("========== event %3d ==========", GetEventCount()));
  for (const TObject* obj = GetFirstInputObject();
       obj!=NULL;
       obj = GetNextInputObject()) {
    AliInfoClass(Form("object: %s", obj->GetName()));
    if (obj->IsA()==AliESDEvent::Class()) {
      // input objects are not supposed to be changed by the component, so they
      // are defined const. However, the implementation of AliESDEvent does not
      // support this and we need the const_cast
      AliESDEvent* esd = dynamic_cast<AliESDEvent*>(const_cast<TObject*>(obj));
      if (esd != NULL) {
        esd->GetStdContent();
        for (Int_t i = 0; i < esd->GetNumberOfTracks(); i++) {
          AliESDtrack* track = esd->GetTrack(i);
          AliInfoClass(Form("---------- track %3d ----------", i));
          track->Print("");
          output.Add(track);
        }
      }
    }
  }

  // publish the array of tracks as output
  PushBack(&output, kAliHLTDataTypeTObjArray|kAliHLTDataOriginSample);

  return 0;
}
```

Event Processing DoEvent

Note

- Component can only use data from the input channel for processing, no global objects
- OCDB obejcts loaded in `DoInit` can provide additional calibration/configuration
- There can be multiple input blocks, each identified with a data type and specification
- *High level* components process ROOT objects as input
  `GetFirstInputObject/GetNextInputObject`
- There is an extended `DoEvent` method which provides direct access to the shared memory *(not covered in the tutorial)*
- Output objects are published with `PushBack`
- There are other base classes for specific component groups, which rename the event processing method
  - `AliHLTTrigger` → `DoTrigger` *(see Trigger component tutorial)*
  - `AliHLTCalibrationProcessor` → `ProcessCalibration`

Cleanup DoDeinit

Overloaded function for component cleanup

```
int AliHLTTutorialComponent::DoDeinit()
{
  // component cleanup, delete all instances of helper classes here

  return 0;
}
```

Runtime configuration

Functions invoked by the framework to announce an update of OCDB objects and configuration

```
int AliHLTTutorialComponent::Reconfigure(const char* cdbEntry, const char* chainId)
{
  // reconfigure the component from the specified CDB entry, or default CDB entry
  HLTInfo("reconfigure '%s' from entry %s", chainId, cdbEntry);

  return 0;
}

int AliHLTTutorialComponent::ReadPreprocessorValues(const char* modules)
{
  // read the preprocessor values for the detectors in the modules list
  int iResult=0;
  TString detectors(modules!=NULL?modules:"");
  HLTInfo("read preprocessor values for detector(s): %s", detectors.IsNull()?"none":detectors.Data());
  return iResult;
}
```