O2 DQ framework tutorial introduction: idea, structure and running

Michael Winn

Department of Nuclear Physics IRFU/CEA, university Paris-Saclay
nearly exclusively based on material by Ionut Arsene (University of Oslo)

O2 Tutorial DQ session, 28.04.2023

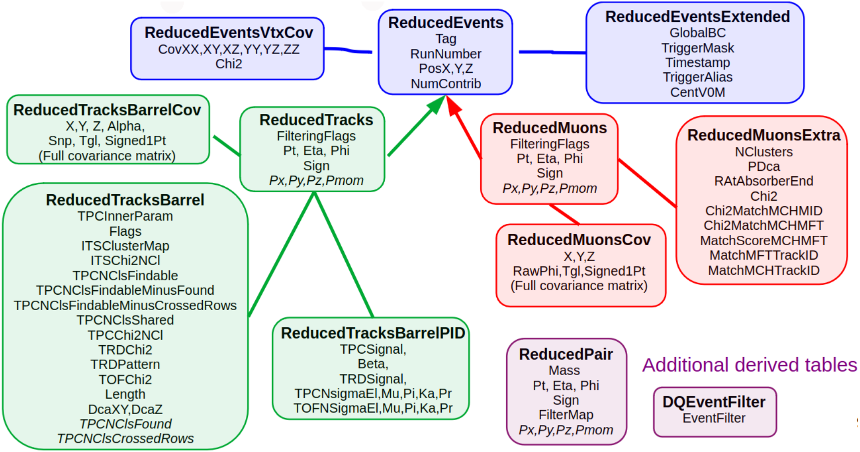
Further detailed material and support

- Today only a short presentation to allow you to work with the framework
- detailed presentation by Ionut and others in DQ meeting in 05/2022 link
- materialin last DQO2 tutorial link
- please subscribe to alice-pwg-DQ-O2@cern.ch
- in case of further questions beyond the tutorial, use mattermost channel: ´O2-DQ Analysis Framework Alpha´

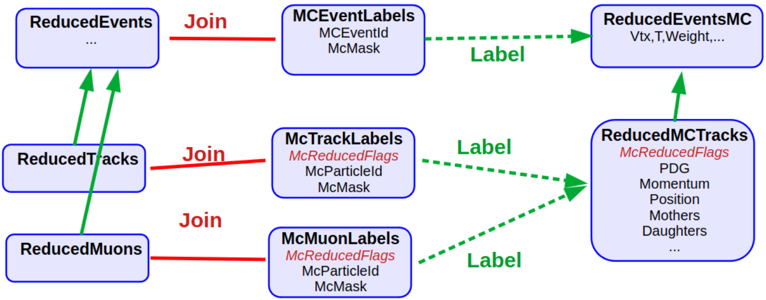
Idea and scope of framework

- Idea: all analyses with single or dileptons (electrons & muons) as part of the analyses
- dilepton + 1/2 track(s) (correlations and b-decays), flow measurements, multiplicity dependence measurements,..
- hence: scope DQ, EM dileptons, single-lepton based HF analyses, could be extended to UD
- realised via modularized derived data
- embedded in a hierarchical running strategy

The O2 DQ Data model



The O2 DQ Data model: MC labelling

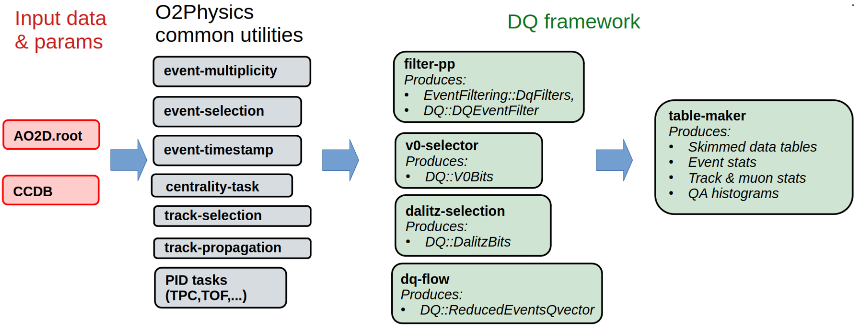


- MC generator level information can be retrieved via MC "labels", joinable to the event, track and muon tables
- Using a skimmed MC model allows for large data size reduction (work with just particles of interest)

Running philosphy

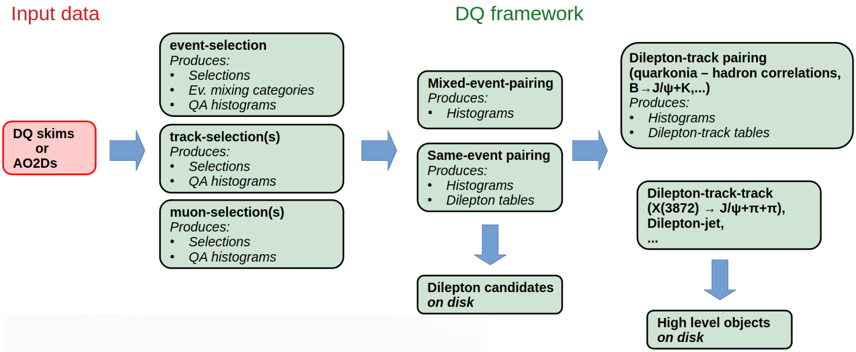
- Run 3: large data volumes implied by increase of luminosity and ALICE high-granularity detectors
- 2-3 levels of data reduction within framework to provide relatively small size output including all information needed for analysis with full luminosity
- trigger layer processing using **filter-pp**: filter-pp-task (software trigger), based on simple, inclusive single and dilepton signatures
- centralised hyperloop for derived data with **table-maker**: reading AO2D, writing reducedAod.root
- user-based running on derived data **table reader**: reading reducedAod.root, writing trees/histograms to analyse
- final output contains candidates and auxiliary sets of tracks (V0 tracks, dalitz leptons)

Skimming Workflow



- Produces a "skimmed and slimmed" data model
- Configurability for
  - selecting events, tracks and muons with multiple parallel selections
  - Amount of event/track/muon information

Analysis workflow



- Runs analysis over DQ skims or Framework/AO2D
- Can replay event/track selections but also use bits based on decisions computed at skimming time
- Produces high level skims for "offline" applications, e.g. machine learning

Structure of an analysis task in code

From PWGDQ/Tasks/tableReader.cxx

```cpp
struct AnalysisTrackSelection {
  Produces<aod::BarrelTrackCuts> trackSel;
  OutputObj<THashList> fOutputList{"output"};
  // The list of cuts should contain all the track cuts needed later in analysis, including
  //  for candidate electron selection (+ eventual prefilter cuts) and other needs like quarkonium - hadron correlations
  // The user must ensure using them properly in the tasks downstream
  // NOTE: For now, the candidate electron cuts must be provided first, then followed by any other needed selections
  Configurable<string> fConfigCuts{"cfgTrackCuts", "jpsiPID1", "Comma separated list of barrel track cuts"};
  Configurable<bool> fConfigQA{"cfgQA", false, "If true, fill QA histograms"};
  Configurable<std::string> fConfigAddTrackHistogram{"cfgAddTrackHistogram", "", "Comma separated list of histograms"};

  HistogramManager* fHistMan;
  std::vector<AnalysisCompositeCut> fTrackCuts;

  void init(o2::framework::InitContext&)
  {
    TString cutNamesStr = fConfigCuts.value;
    if (!cutNamesStr.IsNull()) {
      std::unique_ptr<TObjArray> objArray(cutNamesStr.Tokenize(","));
      for (int icut = 0; icut < objArray->GetEntries(); ++icut) {
        fTrackCuts.push_back(*dqcuts::GetCompositeCut(objArray->At(icut)->GetName()));
      }
    }
    VarManager::SetUseVars(AnalysisCut::fgUsedVars); // provide the list of required variables so that VarManager knows what to fill

    if (fConfigQA) {
      VarManager::SetDefaultVarNames();
      fHistMan = new HistogramManager("analysisHistos", "aa", VarManager::kNVars);
      fHistMan->SetUseDefaultVariableNames(kTRUE);
      fHistMan->SetDefaultVarNames(VarManager::fgVariableNames, VarManager::fgVariableUnits);

      // set one histogram directory for each defined track cut
      TString histDirNames = "TrackBarrel_BeforeCuts;";
      for (auto& cut : fTrackCuts) {
        histDirNames += Form("TrackBarrel_%s;", cut.GetName());
      }

      DefineHistograms(fHistMan, histDirNames.Data(), fConfigAddTrackHistogram); // define all histograms
      VarManager::SetUseVars(fHistMan->GetUsedVars());                           // provide the list of required variables so that VarManager
      knows what to fill
      fOutputList.setObject(fHistMan->GetMainHistogramList());
    }
  }
```

Data tables being produced

Output object (e.g. list of histograms)

Configurables (user input parameters)

Initialization function (run once when the task is created)

Structure of an analysis task in code

From PWGDQ/Tasks/tableReader.cxx

```
template <uint32_t TEventFillMap, uint32_t TTrackFillMap, typename TEvent, typename TTracks>
void runTrackSelection(TEvent const& event, TTracks const& tracks)
{
  VarManager::ResetValues(0, VarManager::kNBarrelTrackVariables);
  // fill event information which might be needed in histograms/cuts that combine track and event properties
  VarManager::FillEvent<TEventFillMap>(event);

  trackSel.reserve(tracks.size());
  uint32_t filterMap = 0;
  int iCut = 0;

  for (auto& track : tracks) {
    filterMap = 0;
    VarManager::FillTrack<TTrackFillMap>(track);
    if (fConfigQA) { // TODO: make this compile time
      fHistMan->FillHistClass("TrackBarrel_BeforeCuts", VarManager::fgValues);
    }

    iCut = 0;
    for (auto cut = fTrackCuts.begin(); cut != fTrackCuts.end(); cut++, iCut++) {
      if ((*cut).IsSelected(VarManager::fgValues)) {
        filterMap |= (uint32_t(1) << iCut);
        if (fConfigQA) { // TODO: make this compile time
          fHistMan->FillHistClass(Form("TrackBarrel_%s", (*cut).GetName()), VarManager::fgValues);
        }
      }
    }
    trackSel(static_cast<int>(filterMap));
  } // end loop over tracks
}

void processSkimmed(MyEvents::iterator const& event, MyBarrelTracks const& tracks)
{
  runTrackSelection<gkEventFillMap, gkTrackFillMap>(event, tracks);
}
void processDummy(MyEvents&)
{
  // do nothing
}

PROCESS_SWITCH(AnalysisTrackSelection, processSkimmed, "Run barrel track selection on DQ skimmed tracks", false);
PROCESS_SWITCH(AnalysisTrackSelection, processDummy, "Dummy function", false);
};
```

"process" function:

- Arguments specify required input data tables
- Frequency of running depends on the input arguments
- Multiple process functions allowed

Process switch: switch on/off tasks / process functions

Generic structure of workflows in O2

- A workflow is a collection of tasks (or DPL devices) running simultaneously in a shared memory environment
- Each task / device must specify:
  – Inputs (data tables, other resources)
  – Outputs (data tables, histograms, etc)
- Can have:
  – Specific initialization function: init()
  – Configurable input parameters: Configurable
  – Other data or function members
- DPL framework organizes/optimizes the chain of running the different tasks based on the specified inputs and outputs

Running a workflow in O2

- Main requirements for the user:
  – Select the needed tasks to be run in the workflow
  – Specify the configuration of each single task / device in the workflow in order to achieve the analysis goals
- Required tasks can be found in the same O2 executable or in different ones
  – Multiple O2 executables can be combined in a pipe:
  o2-analysis1 | o2-analysis2 | ...
- The user must ensure that the workflow can run, i.e. all needed inputs can be read from input files or can be produced by the specified workflow devices

Running a workflow in O2

- Configuring and running a workflow can be a very laborious and error prone process due to many tasks that usually need to be run, so:
  – Task configuration is done using (predefined) .json files, and
  – Workflows are typically run using python scripts
- Example of a command line run without the help of scripts: -
  o2-analysis-dq-table-maker-mc –configuration json://tempConfig.json –severity error –shm-segment-size 12000000000 –aod-writer- json aodWriterTempConfig.json -b | o2-analysis-timestamp –configuration json://tempConfig.json -b | o2-analysis-event-selection – configuration json://tempConfig.json -b | o2-analysis-multiplicity-table –configuration json://tempConfig.json -b | o2-analysis- trackselection –configuration json://tempConfig.json -b | o2-analysis-pid-tof-base –configuration json://tempConfig.json -b | o2- analysis-pid-tof –configuration json://tempConfig.json -b | o2-analysis-pid-tof-full –configuration json://tempConfig.json -b | o2- analysis-pid-tof-beta –configuration json://tempConfig.json -b | o2-analysis-pid-tpc-full –configuration json://tempConfig.json -b | o2- analysis-track-propagation –configuration json://tempConfig.json -b
- Example of command line using a python script:
  - ./runTableMaker.py -runMC –arg internal-dpl-aod-reader:aod-file:AO2D.root configTableMakerMCRun3.json –add_track_prop

A .json configuration file

```json
        "step-value-enumeration": "1",
        "aod-file": "reducedAod_dataBtoJpsiK.root",
        "aod-reader-json": "readerConfiguration_reducedEvent.json"
    },
    "internal-dpl-injected-dummy-sink": "",
    "analysis-event-selection": {
        "cfgMixingVars": "Vtx3",
        "cfgEventCuts": "eventStandardNoINT7",
        "cfgQA": "true",
        "cfgAddEventHistogram": "trigger,cent",
        "processSkimmed": "true",
        "processDummy": "false"
    },
    "analysis-track-selection": {
        "cfgTrackCuts": "jpsiO2MCdebugCuts,kaonPID",
        "cfgDalitzCutId": "32",
        "cfgQA": "true",
        "cfgAddTrackHistogram": "dca,its,tpcpid,tofpid",
        "processSkimmed": "true",
        "processDummy": "false"
    },
    "analysis-muon-selection": {
        "cfgMuonCuts": "muonQualityCuts",
        "cfgQA": "true",
        "cfgAddMuonHistogram": "muon",
        "processSkimmed": "false",
        "processDummy": "true"
    },
    "analysis-event-mixing": {
        "cfgTrackCuts": "jpsiO2MCdebugCuts",
        "cfgMuonCuts": "muonQualityCuts",
        "cfgAddEventMixingHistogram": "barrel,vertexing,flow",
        "cfgMixingDepth": "5",
        "processBarrelSkimmed": "true",
        "processMuonSkimmed": "false",
        "processBarrelMuonSkimmed": "false",
        "processBarrelVnSkimmed": "false",
        "processMuonVnSkimmed": "false",
        "processDummy": "false"
    },
    "analysis-same-event-pairing": {
        "cfgTrackCuts": "jpsiO2MCdebugCuts",
        "cfgMuonCuts": "muonQualityCuts",
        "cfgAddSEPHistogram": "barrel,vertexing,flow",
```

- All tasks in the workflow need to have a specified configuration in .json file - Specify the configurables and process functions
- See e.g. for the track selection task shown in slides 5-6
- N.B. When a task is included in the workflow, at least one process function must be active - Sometime if we want to switch off a task from an executable that is being run, we enable a process function named “processDummy” which does nothing

A .json reader/writer configuration

```json
{
  "InputDirector": {
    "debugmode": true,
    "InputDescriptors": [
      {
        "table": "AOD/REDUCEDEVENT/0",
      },
      {
        "table": "AOD/REEXTENDED/0",
      },
      {
        "table": "AOD/REVTXCOV/0",
      },
      {
        "table": "AOD/REQVECTOR/0",
      },
      {
        "table": "AOD/REDUCEDTRACK/0",
      },
      {
        "table": "AOD/RTBARREL/0",
      },
      {
        "table": "AOD/RTBARRELCOV/0",
      },
      {
        "table": "AOD/RTBARRELPID/0",
      },
      {
        "table": "AOD/RTMUON/0",
      },
      {
        "table": "AOD/RTMUONEXTRA/0",
      },
      {
        "table": "AOD/RTMUONCOV/0",
      },
      {
        "table": "AOD/AMBIGUOUSTRACK/0",
      },
      {
        "table": "AOD/AMBIGUOUSFWDTR/0",
      },
      {
        "table": "AOD/DALITZBITS/0",
      }
    ]
  }
}
```

- A reader/writer configuration file specifies the data tables to be read from or written to disk
  - Needed when one works with data models other than the central O2/Framework model
- The file can specify just the table identifier or it can specify detailed information, e.g. customized names for each data member in a table
- These files are typically needed when:
  - Analyzing skimmed data - Writing skimmed data to disk

The goals of this tutorial

- Understand the way workflows are configured
- Run data and MC skimming, tailored to your analysis
  – Merge the skims into larger data frames (to avoid empty or nearly empty data frames)
- Analyze the skims
  – Dilepton analysis
  – Dilepton + hadron analysis
  – Run over MC and match reconstructed and generator level objects