CxAOD structure and CxAODReader

Daniel Büscher

Universität Freiburg

September 23, 2015

Introduction

- Discussing the object retrieval and systematic loop in CxAODReader
- Showing example code from SVN (links below), but simplified in many places

$\Rightarrow$ The SVN code has more checks and optimizations, which should be kept

Code from:

`https://svnweb.cern.ch/trac/atlasoff/browser/PhysicsAnalysis/HiggsPhys/Run2/Hbb/CxAODFramework/CxAODReader/`
`https://svnweb.cern.ch/trac/atlasphys-exa/browser/Physics/Exotic/Analysis/DibosonResonance/Data2015/Code/CxAODFramework_DB/CxAODReader_DB/`

Basic info on the xAOD EDM:

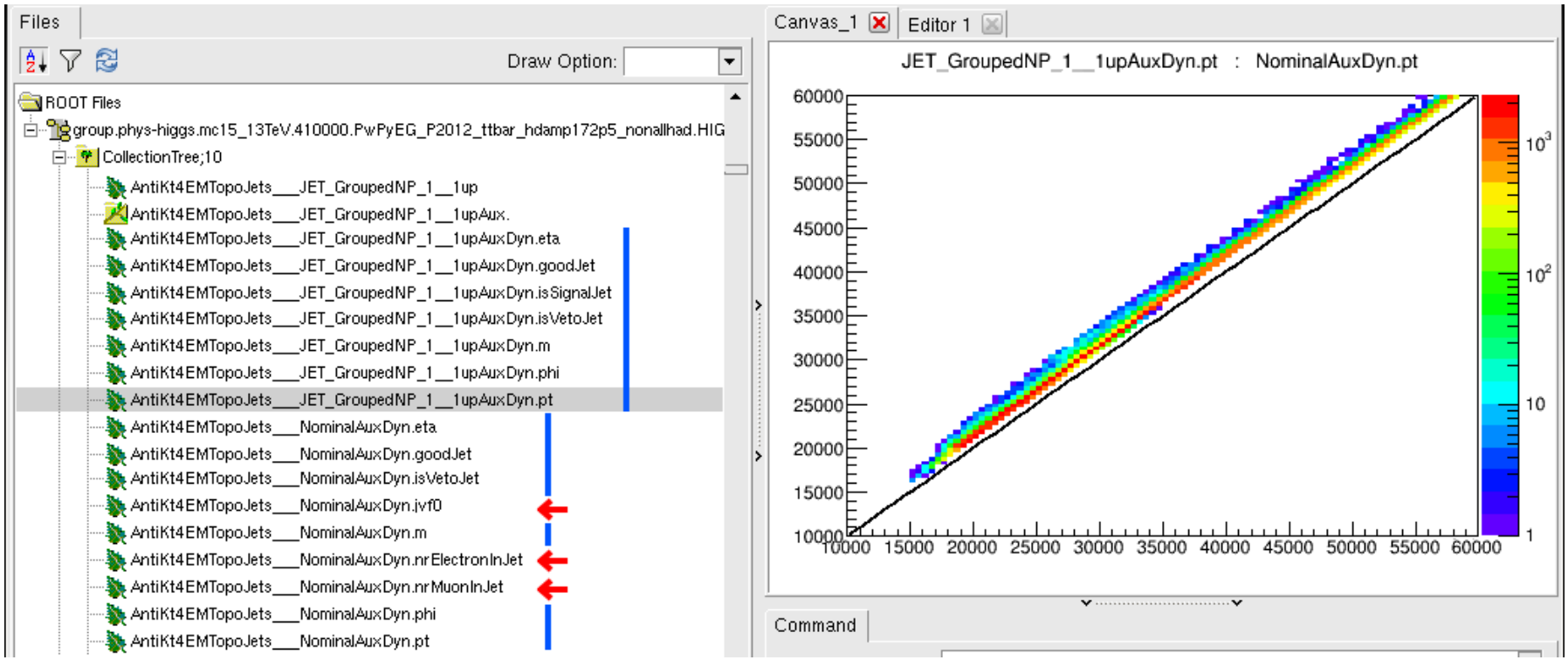
`https://twiki.cern.ch/twiki/bin/view/AtlasComputing/SoftwareTutorialxAODAnalysisInROOT`

Tutorial slides with more info on CxAOD structure:

`https://svnweb.cern.ch/trac/atlasoff/export/HEAD/PhysicsAnalysis/HiggsPhys/Run2/Hbb/CxAODFramework/FrameworkSub/trunk/doc/15-02-17_tutorial/CxAODtutorial.pdf`

CxAOD structure

- Systematic variations of objects (particles) are stored as shallow copies (SCs) of Nominal
  - Only the Nominal collection contains all branches
  - SCs contain only those branches affected by systematics
  - $\Rightarrow$ Advantage in disk size and CPU time over conventional Ntuples
- Example for jets:
  - Four-vectors and selection flags are stored for Nominal as well as for the variation
  - "jvf0" and "nrElectron/MuonInJet" are only stored for Nominal jets
  - Note: CxAODs provide an easy drawing of systematic effects!



xAOD EDM and implications

- At first look the structure of the CxAOD seems rather complicated
- However,the xAOD EDM provides convenient way to read variations:
  - SCs of objects can be used exactly as the Nominal ones

  $\Rightarrow$ Both can be accessed trough e.g. xAOD::Jet
  - If one tries to read a non-exising branch of the SC the Nominal value is returned
- This implies that additional objects have to be stored
  - If an object is stored for some variation it has to be stored for all variations (e.g. if a jet passed selection for JES-up, it is stored for Nominal, JES-do, ...)

  $\Rightarrow$ The CxAOD can contain objects that did not actually pass the pre-selection
  - The same applies to events

$\Rightarrow$ Object and event pre-selection have to be re-applied when reading the CxAOD!
  - Objects can be re-selected using flags + re-applying overlap removal
  - For event selection the same routine can be re-applied

  $\Rightarrow$ Both implemented by CxAODReader

Initialize selection in CxAODReader

- AnalysisReader::initizializeSelection() determines which event selection class is used

⇒ Should be the same as for producing the CxAOD

- In addition, a "fill function" is defined: fills histograms, output trees, ...

```
1  EL::StatusCode AnalysisReader_DB :: initializeSelection() {
2    m_config->getif<int>("analysisType", m_analysisType);
3    if (m_analysisType==VH1LepHist) {
4      m_eventSelection = new lvqq_MainEvtSelection(m_config);
5      m_fillFunction = std::bind(&AnalysisReader_DB::fill_VH1LepHist, this);
6    }
7    // initialize other selections, MVATree, ...
8  }
```

The event loop in CxAODReader

- For each event a loop on the variations is performed (different to most Run 1 codes)
  - **(a)** ObjectReader take care of reading the xAOD containers (via TEvent::retrieve())
  - **(b)** The overlap removal and event selection are applied
  - **(c)** A user defined fill function is called
- Note: (b), (c) don't need to know which variation is processed, they are just provided with objects of a given variation

```
1  EL::StatusCode AnalysisReader::execute () {
2    for (std::string varName : m_variations) {
3      m_electrons = m_electronReader->getObjects(varName);
4      m_jets      = m_jetReader->getObjects(varName);
5      // ... retrieve other containers ...
6      setObjectsForOR(m_electrons, ...);
7      m_overlapRemoval->removeOverlap(m_electrons, ...)
8      bool passSel = m_eventSelection->passPreSelection(m_electrons, ...);
9      passSel     &= m_eventSelection->passSelection(m_electrons, ...);
10     if (!passSel) continue;
11     m_fillFunction();
12   }
13 }
```

The fill function

- The fill function retrieves the result of the event selection

⇒ The result contains the selected objects for the present variation

- One could directly access m_jets, but these are not guaranteed have passed selection!
- The xAOD objects provide methods retrieving four-vectors and some other properties
- Other values can be read using PROPs (defined in CommonProperties.h or anywhere else)

⇒ Analysis variables can be calculated and filled into histograms, ...

```
1  EL::StatusCode AnalysisReader_DB::fill_VH1LepHist() {
2    DBResult selectionResult = ((lvqq_MainEvtSelection*)m_eventSelection)->result();
3    std::vector<const xAOD::Electron*> electrons = selectionResult.el;
4    std::vector<const xAOD::Jet*>      largeJets = selectionResult.largeJets;
5    std::vector<const xAOD::Jet*>      trackJets = selectionResult.trackJets;
6    // ... retrieve other containers ...
7    TLorentzVector lepVec  = electron.at(0)->p4();
8    TLorentzVector higgVec = largeJets.at(0)->p4();
9    float btagVal          = Props::MV2c20.get(trackJets.at(0));
10   // ... compute analysis variables, fill histograms, output tree ...
11 }
```

Summary

Summary

- The CxAOD format provides an efficient way to store and read systematic variations
- Not all stored objects/events are guaranteed to have passed pre-selection

$\Rightarrow$ Selection has to be re-applied!

- This is implemented by CxAODReader
  - Provides an efficient systematics loop
  - Final analysis variables can be easily calculated using xAOD objects