Tutorial on (HSG5) Analysis Framework for Run 2

Kristian Gregersen, Josh McFayden

Outline

- **Framework Layout**
  - How the code is organised
  - What functionality/calibration/etc is included
  - Contents of the output file
  - Configuration
- **Demonstration of CxAODMaker (Interactive)**
  - The "Quick Guide"
  - Browse output xAOD in ROOT
- **Grid running**
- **How to modify the code**
  - How to add new variables to the output
  - How to add new analysis code
- **How to get help**
  - Useful links / mailing lists

DISCLAIMER

- This framework is developed within the VHbb analysis group (HSG5).
  “Out of the box” compile/runtime configurations follow the VHbb analysis!

- **Usage**:
  - For running the VHbb analysis, the code is simply checked out and run.
  - For optimisation studies of the VHbb analysis, the code is checked out and modified. Optimisations are then merged into the repository.
  - For *other analysis*, the code is checked out and used as the base for new derived analysis classes. More on this later…

- **“Out of the box”-contents:**
  - The recommended calibrations are applied (CP tools).
  - Overlap removal is performed (using official ASG tool).
  - MET is rebuild.
  - Object and event selections are applied.
  - Systematic variations are run, i.e. the above steps are rerun for each variation.
  - The output xAOD file contains the superset of objects among sys. variations, along with event level information.

Framework layout

- Based on RootCore (a `C++` compilation framework which depends on `ROOT`).
- Uses EventLoop (EL) and SampleHandler.
- Currently there are 7 "RootCore" packages :

`CxAODMaker`	Contains the main code for processing the input xAOD files, applying the calibration tools (CP tools), and writing the output CxAOD (C for Calibrated).
`CxAODTools`	Contains tools that are shared outside of the main CxAODMaker work-flow, such as object and event selection.
`FrameworkExe`	Contains the code for the executables for running CxAODMaker jobs, along with any associated configuration files.
`FrameworkSub`	Contains files relating to defining datasets and related information for tracking processing.
`CxAODReader`	Contains code to read CxAOD using the xAOD EDM (Event Data Model). Implements plotting style used in HSG5.
`TupleMaker`	Contains example code for producing a compact flat TTree based tuple that could be used for diagnostics or analysis.
`TupleReader`	Contains the common TTree definition and example code to read a compact flat TTree.

Framework layout

- Based on RootCore (a `C++` compilation framework which depends on `ROOT`).
- Uses EventLoop (EL) and SampleHandler.
- Currently there are 7 "RootCore" packages :

`CxAODMaker`

`CxAODTools`

`FrameworkExe`

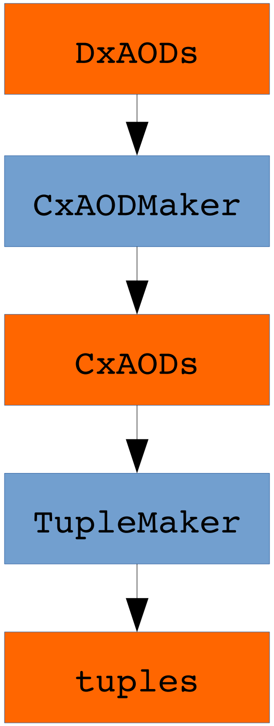
`FrameworkSub`

`CxAODReader`

`TupleMaker`

`TupleReader`

Focus will be on these today



CxAODMaker

<Particle>Handler

- <Particle> is:
  Electron, FatJet, Jet, Muon, Photon, Tau, TrackJet, TruthParticle.
- Applies calibrations (where appropriate) including sys. variations.
- Classifies particles and attaches corresponding flags to each particle for each sys. variation (HSG5 specific, e.g. 'isVHLooseElectron').

EventInfoHandler

- Holds event level info, e.g. run/event number, GRL pass/fail, MC event weight, vertex info,...
- Visible to Particle Handlers

<u>Note:</u> Trigger info is still not accessible in current xAODs (JIRA)

METHandler

- Rebuilds the MET from calibrated objects, including sys. varations.
- Uses 'official' METRebuilder tool (which implements its own overlap removal.)

AnalysisBase

- Controls the work flow.
- Defines the EL algorithm.
- Contains the methods executed by EL, i.e. initialize(), execute(),...
- Declares and executes all “handlers” and the “selector”, i.e. these are plugged in to the EL work-flow through this class.
- Has access to the TEvent, and pass it on to the Handlers

EventSelector

- Applies event level selection.
- Uses 'official' overlap removal tool for resolving conflicts for object appearing in multiple containers.
- Apart from the overlap removal, the selection is HSG5 specific.

Note: Each Handler retrieves/writes data from/to TEvent directly.

CxAODMaker

- **List of Calibration Tools in framework**
  - Jets :
    - JetCalibrationTool
    - JetCleaningTool
    - JERTool
    - JERSmearingTool
    - BTaggingEfficiencyTool
  - Taus :
    - TauSmearingTool
    - TauSelectionTool
    - TauEfficiencyCorrectionsTool
  - Photons :
    - EgammaCalibrationAndSmearingTool
    - AsgPhotonIsEMSelector
  - Electrons :
    - EgammaCalibrationAndSmearingTool
    - AsgElectronLikelihoodTool
    - AsgElectronIsEMSelector
  - Muons :
    - MuonCalibrationAndSmearingTool
    - MuonSelectionTool
    - MuonEfficiencyScaleFactors

- **ASG analysis tools xAOD migration progress**
  - https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/ASGUserAnalysisToolsxAODMigration

CxAODMaker

- **Contents of the output xAOD**

- The main guideline is to store as little as possible – which in practical terms implies a trade-off :

  - The output file is (fairly) well organised, without too many variables cluttering/creating confusion.
  - But, occasionally, this will also result in important variables missing, and thus re-running of the code.

- For particle-type objects:
  - The 4-vector is always stored.
  - Other chosen quantities are explicitly added by hand (with 'decorators') and stored. (e.g. object scale factors, or flags which specify if an object passed some selection criteria).
  - The particle-index is also stored, i.e. the index of a given particle in the container in the input file. This enables exact comparisons of objects if/when discrepancies occur between analysis groups (instead of e.g. doing 4-vector comparisons to figure out which objects match up).

- Event information:
  - Run number, event number, NVtx3Trks, MCEventWeight, …
  - Eventually, also trigger information.

- Systematic variations are stored with the original collection name as "base" name and then the name of the variation appended to it (after 3 "_"), so e.g "Muons___Nominal" and "Muons___someVariation".

CxAODMaker

- **Configuration**

- The configuration file is located here:
  FrameworkExe/data/CxAODMaker-job.cfg

```
# This is a ConfigStore file, please see ConfigStore.cxx for info.

bool debug                     = false

bool printObjectCounts         = false
bool printKnownVariations      = false

bool applyObjectSelection      = true
bool applyOverlapRemoval       = true

bool applyEventSelection       = true
string selectionName           = 1lep
bool autoDetermineSelection    = false

# note: container names can be commented out to disable their processing

string muonContainer           = Muons
string electronContainer       = ElectronCollection
# no taus in HIGG5 derivations
#string tauContainer            = TauRecContainer
string photonContainer         = PhotonCollection
string jetContainer            = AntiKt4LCTopoJets
string jetAlgoName             = AntiKt4LCTopo
# JetUncertaintiesTool does not like AntiKt4TopoEM
#string jetSpectatorContainer   = AntiKt4EMTopoJets
#string jetSpectatorAlgoName    = AntiKt4TopoEM
string fatJetContainer         = CamKt12LCTopoJets
#string trackJetContainer       = AntiKt4ZTrackJets
string METContainer            = MET_RefFinal
string truthParticleContainer  = TruthParticle
string truthVertexContainer    = TruthVertex
string grl = $ROOTCOREBIN/data/CxAODMaker/data12_8TeV.periodAllYear_DetStatus-v61-pro14-02_DQDefects-00-01-00_PHYS_StandardGRL_All_Good.xml

# "Nominal" is always written
vector<string> variations      = EG_SCALE_ALL MUONS_SCALE MUONS_ID MUONS_MS
vector<string> weightVariations = MUONSFSYS
vector<string> variationsSymm = JetEResol
bool storeOriginal             = true
```

Note: By default the VHbb "1 lepton" selection is applied

Names of collections, used by MuonHandler, JetHandler,...

List of sys. variations

TupleMaker

- **Produces flat ntuples**
  - int, float and arrays
  - An EventLoop algorithm, like AnalysisBase in CxAODMaker.
  - Configured independent of CxAODMaker: `FrameworkExe/data/TupleMaker-job.cfg`

```
# This is a ConfigStore file, please see ConfigStore.cxx for info.

string TupleMaker.PhotonsIn       = PhotonCollection
string TupleMaker.MuonsIn         = Muons
string TupleMaker.ElectronsIn     = ElectronCollection
string TupleMaker.JetsIn          = AntiKt4LCTopoJets
string TupleMaker.FatJetsIn       = CamKt12LCTopoJets
string TupleMaker.MetIn           = MET_RefFinal
string TupleMaker.Label           = tuple

vector<string> TupleMaker.Variations = Original Nominal
string         TupleMaker.VariationMode = block
bool           TupleMaker.UseEventInfo = true
```

Names of collections (sys. variations are appended automatically)

List of sys. variations

- Run with : `hsg5frameworkTuple`
  - will run CxAODMaker followed by TupleMaker such that the output collections in TEvent from the CxAODMaker are read in by TupleMaker.

- The systematic variations can be written out in 3 different ways:
  - "file" : A separate file with one TTree for each variation
  - "tree" : One file with one TTree for each variation
  - "block" : One file with one tree where the variables have the variation appended to their name

FrameworkExe

- **The framework executables**

  - Contains the files that defines the "main()" methods which are compiled into executables
    - `hsg5framework        (runs CxAODMaker)`
    - `Hsg5frameworkTuple   (runs CxAODMaker and TupleMaker in sequence)`
    - `Hsg5frameworkReader  (runs CxAODMakerReader, which processes CxAODs)`

  - Configured in
    `FrameworkExe/data/framework-run.cfg`

```
# This is a ConfigStore file, please see ConfigStore.cxx for info.

int maxEvents                 = -1
string CxAODConfig            = data/FrameworkExe/CxAODMaker-job.cfg
string TupleConfig            = data/FrameworkExe/TupleMaker-job.cfg
string COMEnergy              = 13TeV

###################
# for grid only -> it is not affecting interactiv run
###################
# tag version
string vtag                   = p1_141216
# test job
bool submit                   = false
# exlcude site
string excludedSite           = none
#string excludedSite           = ANALY_ARNES,ANALY_LUNARC,ANALY_SiGNET,ANALY_RAL_SL6,ANALY_BNL_SHORT,ANALY_DCSC,ANALY_NSC,ANALY_UIO
# lot of files per job
double nGBPerJob              = 8
double nFilesPerJob           = -1
# 1 file only per job
#double nGBPerJob              = -1
#double nFilesPerJob           = 1

```

Reads in the config files for CxAODMaker and TupleMaker from the previous 2 slides by default.

  - Final note on runtime configuration:
    - The steering file is distributed to all the Handlers in CxAODMaker, so modifying/adding configuration variables to a Handler is easy.

Demonstration of CxAODMaker / TupleMaker

Quick Start

For a quick setup of the framework please log into a clean lxplus session. Support for local machines is planned. Then you can copy&paste the following script to check out and compile the code:

```
# setup ATLAS environment
setupATLAS

# make some working direcory
mkdir CxAODFramework
cd CxAODFramework/

# setup RootCore
rcSetup Base,2.0.23
# Note, from Base,2.0.16 and later there are a lot of compile time warnings concerning the boost lib
# which are due to a new compile flag in ROOT, -Wunused-local-typedefs

# check out CxAODFramework
svn co svn+ssh://svn.cern.ch/reps/atlasphys/Physics/Higgs/HSG5/software/VHAnalysis/LHCRun2/CxAODFramework/CxAODMaker/trunk CxAODMaker
svn co svn+ssh://svn.cern.ch/reps/atlasphys/Physics/Higgs/HSG5/software/VHAnalysis/LHCRun2/CxAODFramework/CxAODReader/trunk CxAODReader
svn co svn+ssh://svn.cern.ch/reps/atlasphys/Physics/Higgs/HSG5/software/VHAnalysis/LHCRun2/CxAODFramework/CxAODTools/trunk CxAODTools
svn co svn+ssh://svn.cern.ch/reps/atlasphys/Physics/Higgs/HSG5/software/VHAnalysis/LHCRun2/CxAODFramework/FrameworkExe/trunk FrameworkExe
svn co svn+ssh://svn.cern.ch/reps/atlasphys/Physics/Higgs/HSG5/software/VHAnalysis/LHCRun2/CxAODFramework/FrameworkSub/trunk FrameworkSub
svn co svn+ssh://svn.cern.ch/reps/atlasphys/Physics/Higgs/HSG5/software/VHAnalysis/LHCRun2/CxAODFramework/TupleMaker/trunk TupleMaker
svn co svn+ssh://svn.cern.ch/reps/atlasphys/Physics/Higgs/HSG5/software/VHAnalysis/LHCRun2/CxAODFramework/TupleReader/trunk TupleReader

# scan packages and compile
rc find_packages
rc compile
```

A test job can be run with:

```
hsg5framework
```

FrameworkExe

- **Grid running**
  - We will not cover this part of the framework in this tutorial, but refer to the documentation provided here: https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/CxAODFramework

Grid running

For setting up the grid log into a clean lxplus session and run in your working direcory:

```
setupATLAS
localSetupDQ2Client --skipConfirm # just for dq2-get -> no need for job submission
voms-proxy-init -voms atlas
localSetupPandaClient --noAthenaCheck
rcSetup
```

Find the list of samples in FrameworkSub /In, comment non-desired samples to run on and execute (for example) :

```
hsg5frameworkTuple FrameworkSub/In/list_sample_grid.*TeV.AOD.txt
```

The code scans the argument for "grid" to change to the grid driver

Modifying the code

- **Adding a variable to the output**
  - Output variables are set in the method "setVariables()" which each handler has, e.g. ElectronHandler:

```
EL::StatusCode ElectronHandler::setVariables(xAOD::Electron * inElectron, xAOD::Electron * outElectron, bool isSysVar)
{

  //TODO clean up
  //TODO add check if variables changed due to calibrations?

  // set four momentum
  setP4( inElectron , outElectron );

  //set isolation
  outElectron->setIsolationValue(inElectron->isolationValue(xAOD::Iso::ptcone20),xAOD::Iso::ptcone20);
  //outElectron->setIsolationValue(inElectron->isolationValue(xAOD::Iso::topoetcone30_corrected),xAOD::Iso::topoetcone30_corrected);
  outElectron->setIsolationValue(inElectron->isolationValue(xAOD::Iso::topoetcone30),xAOD::Iso::topoetcone30);

  // set something without having a pre-defined method
  m_decorator.copy(inElectron, outElectron, ElecIntProps::isVeryTightLH);
  m_decorator.copy(inElectron, outElectron, ElecIntProps::isVHLooseElectron);
  m_decorator.copy(inElectron, outElectron, ElecIntProps::isZHSignalElectron);
  m_decorator.copy(inElectron, outElectron, ElecIntProps::isWHSignalElectron);
```

- The I/O structure in the code is not entirely transparent – the "inElectron" is an object from a socalled "shallow copy" of the collection in the input file. This is the calibrated object which has additional "decorations" attached to it. The "outElectron" is the object which is written out. For the nominal calibration, this is a new memory allocation, while for sys. variations, it is a shallow copy of the nominal output object.

Modifying the code

- **Adding a variable to the output**
  - Output variables are set in the method “setVariables()” which each handler has, e.g. ElectronHandler:

```
EL::StatusCode ElectronHandler::setVariables(xAOD::Electron * inElectron, xAOD::Electron * outElectron, bool isSysVar)
{

  //TODO clean up
  //TODO add check if variables changed due to calibrations?

  // set four momentum
  setP4( inElectron , outElectron );

  //set isolation
  outElectron->setIsolationValue(inElectron->isolationValue(xAOD::Iso::ptcone20),xAOD::Iso::ptcone20);
  //outElectron->setIsolationValue(inElectron->isolationValue(xAOD::Iso::topoetcone30_corrected),xAOD::Iso::topoetcone30_corrected);
  outElectron->setIsolationValue(inElectron->isolationValue(xAOD::Iso::topoetcone30),xAOD::Iso::topoetcone30);

  // set something without having a pre-defined method
  m_decorator.copy(inElectron, outElectron, ElecIntProps::isVeryTightLH);
  m_decorator.copy(inElectron, outElectron, ElecIntProps::isVHLooseElectron);
  m_decorator.copy(inElectron, outElectron, ElecIntProps::isZHSignalElectron);
  m_decorator.copy(inElectron, outElectron, ElecIntProps::isWHSignalElectron);
```

- So, what this method does is to copy properties from the calibrated object (“inElectron”) to the output object (“outElectron”).

- **Thus, to add a variable to “outElectron”, a line must be put here copying the property**.

Modifying the code

- **Adding a variable to the output**
  - In case **the variable is *not* a known property** to "inElectron", it must first be calculated and added to this object, e.g. in the method decorate():

```
EL::StatusCode ElectronHandler::decorate(xAOD::Electron * electron)
{

  //selection tools
  //---------------
  //retrieve decision from tools and decorate electron with it
  //perform actual selection later
  int isVeryLooseLH = static_cast<int>(m_checkVeryLooseLH->accept(electron));
  int isVeryTightLH = static_cast<int>(m_checkVeryTightLH->accept(electron));
  int isTightPP     = static_cast<int>(m_ElectronIsEMSelector->accept(electron));
  m_decorator.set(electron, ElecIntProps::isVeryLooseLH, isVeryLooseLH);
  m_decorator.set(electron, ElecIntProps::isVeryTightLH, isVeryTightLH);
  m_decorator.set(electron, ElecIntProps::isTightPP, isTightPP);

  return EL::StatusCode::SUCCESS;

}
```

This is the same object as "inElectron" on the previous slide

  - But, there is more...

Modifying the code

- **Adding a variable to the output**
  - In case the property is unknown to the object, one has to add it to its decorator which is defined in CxAODTools, e.g. for electrons: CxAODTools/CxAODTools/ElectronDecorator.h

```
#ifndef CxAODTools_ElectronDecorator_H
#define CxAODTools_ElectronDecorator_H

#include "ObjectDecorator.h"

enum class ElecBoolProps {
  // OR tool
  overlaps,
  passPreSel,
};

enum class ElecIntProps {
  // common stuff
  partIndex,
    passOR,
    passORGlob,
    // e-gamma ID
    isVeryLooseLH,
    isVeryTightLH,
    isTightPP,
    isMediumPP,
    isLoosePP,
    // analysis quality assignment
    isVHLooseElectron,
    isVHSignalElectron,
    isZHSignalElectron,
    isWHSignalElectron,
    isTTLHOLRElectron,
    isTTLHDiLepVetoElectron,
    isTTLHPreSelElectron,
    isTTLHIsolElectron,
    isTTLHSoftElectron,

};

enum class ElecFloatProps { IDEff, IDEffSys };
```

Boolean properties

Integer properties

Float properties

  - That should do it

Modifying the code

- **Doing a different analysis than VHbb**
  - This case is a bit more involved, and there are several ways to do it.
  - We have experimented with this, and some volunteers have already tried it out. Here's a link to the TWiki with the current tutorial: CxAODFrameworkTutorial

Implementing a new analysis

Implementing a new analysis in this framework is done by deriving new classes of the existing ones. This provides the ability to replace basically any piece of code you like. However, one has to be careful... TODO elaborate

Creating new packages

```
rc make_skeleton CxAODMaker_Tutorial
rc make_skeleton CxAODTools_Tutorial
rc make_skeleton FrameworkExe_Tutorial
```

CxAODMaker _Tutorial/cmt/Makefile.RootCore

```
PACKAGE_DEP      = CxAODMaker CxAODTools_Tutorial
```

CxAODTools _Tutorial/cmt/Makefile.RootCore

```
PACKAGE_DEP      = CxAODTools
```

FrameworkExe _Tutorial/cmt/Makefile.RootCore

```
PACKAGE_DEP      = FrameworkExe CxAODMaker_Tutorial
```

Compile

```
rc find_packages
rc compile
```

How to get help

- Ask Josh or Kristian 🙂
- Framework mailing list:
  atlas-phys-higgs-hsg5Framework
- Framework TWiki:
  CxAODFramework
- Framework repository:
  Physics/Higgs/HSG5/software/VHAnalysis/LHCRun2/CxAODFramework
- xAOD Tutorial:
  SoftwareTuturialxAODAnalysisInROOT
- RootCore Twiki:
  RootCore