CxAODFramework tutorial

Daniel Büscher, David Jamin
for the Hbb group

Universität Freiburg

February 15, 2015

Outline

Outline

- Introduction to the framework
- ⇒ Session 1: setting up the code, run a test job
- Details on CxAODMaker
- ⇒ Session 2: adding a variable, read the output
- Details on CxAODReader
- ⇒ Session 3: adding histograms, making plots
- How to implement a new analysis
- ⇒ Session 4: adding packages and classes

Links and mailing list

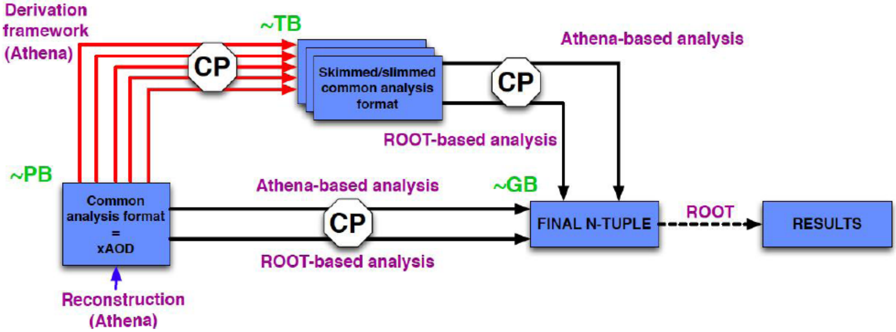
- This tutorial builds up on the xAOD tutorial
  https://twiki.cern.ch/twiki/bin/view/AtlasComputing/SoftwareTutorialxAODAnalysisInROOT
- Framework TWiki
  https://twiki.cern.ch/twiki/bin/view/AtlasProtected/CxAODFramework
- Framework repository
  https://svnweb.cern.ch/trac/atlasphys/browser/Physics/Higgs/HSG5/software/VHAnalysis/LHCRun2/CxAODFramework
- HSG5 Framework mailing list: atlas-phys-higgs-hsg5Framework

Thanks Kristian for providing slides!

https://indico.cern.ch/event/373130/session/1/contribution/9/material/slides/0.pdf

Introduction to the framework

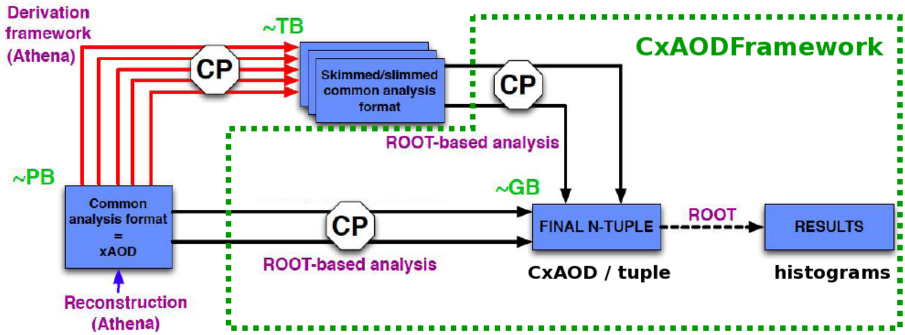
Run 2 workflow



Run 2 workflow

- adapt the new analysis scheme given by the ASG

Run 2 workflow



Run 2 workflow

- adapt the new analysis scheme given by the ASG
- the CxAODFramework can process xAODs and DxAODs
- pruduce a small CxAOD ("C" for calibrated) or a Ntuple
- futher tools for reading the CxAOD, filling histograms and plotting are in place

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

Framework layout

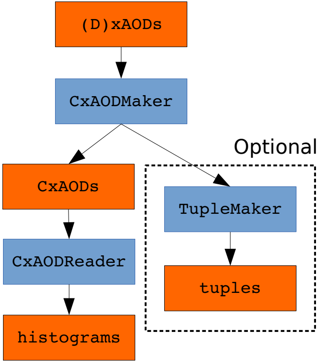
- Based on RootCore (a C++ compilation framework which depends on ROOT).
- Uses EventLoop (EL) and SampleHandler.
- Currently there are 7 RootCore packages:

`CxAODMaker`	Main code for processing the input xAOD files, applying the calibration tools (CP tools), and writing the output CxAOD.
`CxAODTools`	Contains tools that are shared outside of the main CxAOD-Maker work-flow, such as object and event selection.
`FrameworkExe`	Contains the code for the executables for running CxAOD-Maker 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`



CxAODMaker layout

`<Object>Handler`

- `<Object>` 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').

Note: “Object” is mostly for “physics object” or “particle”

`EventInfoHandler`

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

`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

Note: Trigger info is still not accessible in current xAODs (JIRA)

`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.

Calibration Tools

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

Contents of the output CxAOD

The main guideline is to store as little as possible, which implies a trade-off

- The output file is (fairly) well organised, without too many variables creating confusion.
- Occasionally, this will also result in 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 for some selection criteria).
- The index of each particle in the input is also stored

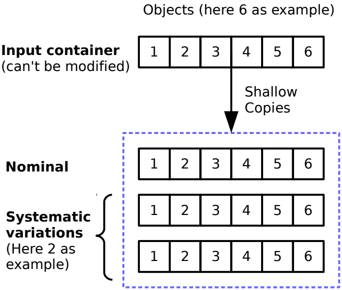
Event information

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

Systematic variations

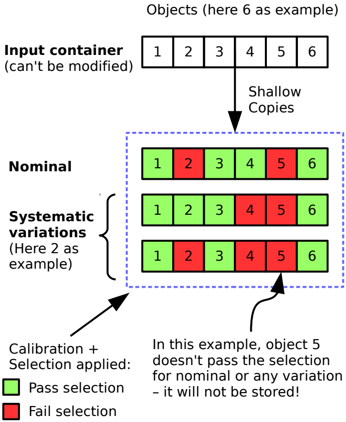
- Stored with the original collection name plus the name of the variation after 3 "_". E.g "Muons___Nominal" and "Muons___someVariation".
- The uncalibrated collections are stored as well as "___Original"

Details of the I/O structure



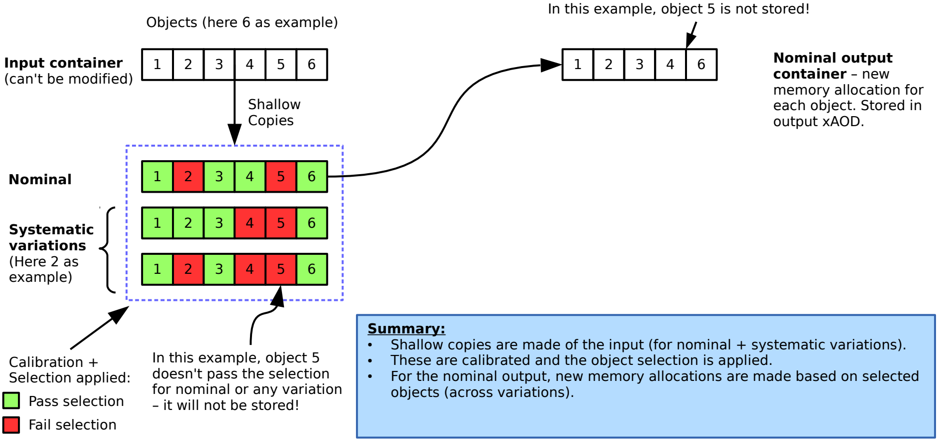
Summary:
- Shallow copies are made of the input (for nominal + systematic variations).

Details of the I/O structure



Summary:
- Shallow copies are made of the input (for nominal + systematic variations).
- These are calibrated and the object selection is applied.

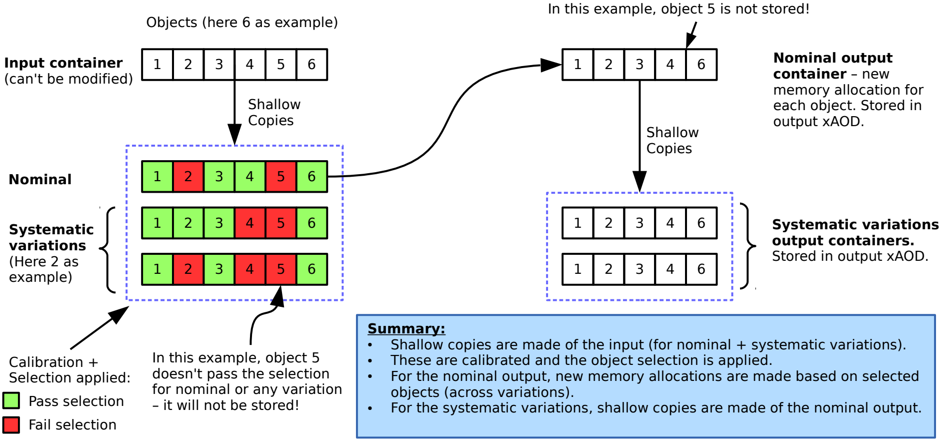
Details of the I/O structure



Summary:

- Shallow copies are made of the input (for nominal + systematic variations).
- These are calibrated and the object selection is applied.
- For the nominal output, new memory allocations are made based on selected objects (across variations).

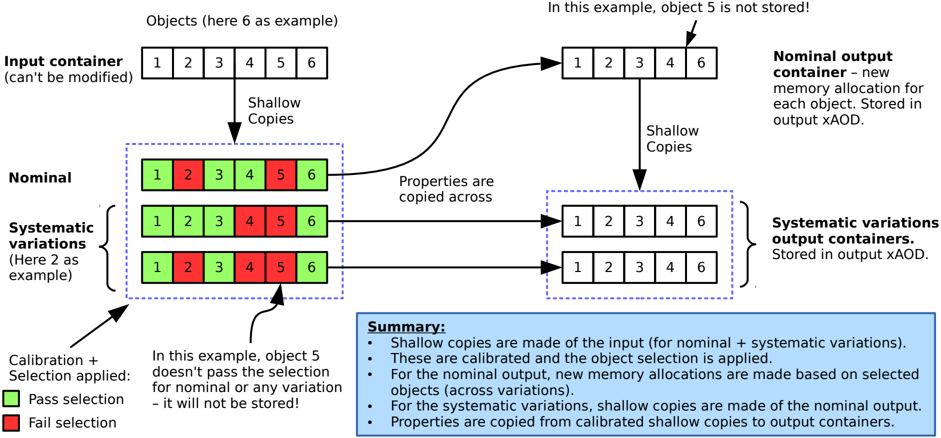
Details of the I/O structure



Summary:

- Shallow copies are made of the input (for nominal + systematic variations).
- These are calibrated and the object selection is applied.
- For the nominal output, new memory allocations are made based on selected objects (across variations).
- For the systematic variations, shallow copies are made of the nominal output.

Details of the I/O structure



Summary:

- Shallow copies are made of the input (for nominal + systematic variations).
- These are calibrated and the object selection is applied.
- For the nominal output, new memory allocations are made based on selected objects (across variations).
- For the systematic variations, shallow copies are made of the nominal output.
- Properties are copied from calibrated shallow copies to output containers.

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

List of sys. variations

TupleMaker

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 in one job
- the output collections from CxAODMaker are passed through TEvent to TupleMaker
- both, CxAOD and tuple, are written out

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, variables have the variation appended to their name

Note that some information might be lost from CxAOD to tuple

FrameworkExe

FrameworkExe

- 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

⇒ modifying/adding configuration variables to a Handler is easy.

Session 1

- Check out and compile the code
- Run CxAODMaker and look at the output

Details on CxAODMaker

CxAODMaker workflow

```
AnalysisBase : EL::Algorithm

histInitialize()    1/job
```

AnalysisBase

- AnalysisBase is an EventLoop algorithm

⇒ has pre-defined methods that are called in the job

- histInitialize() books 1 histogram: "MetaData_EventCount":
  holding event counts and sum of weights from the job and from DxAOD meta data
- no other histograms (and no cut-flow counter) currently

CxAODMaker workflow

AnalysisBase : EL::Algorithm	
histInitialize()	1/job
fileExecute()	1/file

AnalysisBase

- fileExecute() loads meta data from DxAOD and prints some basic info
- note that event loop seems to call the first fileExecute() before histInitialize()

$\Rightarrow$ we have a hack in place to ensure the desired order (histInitialize() first)

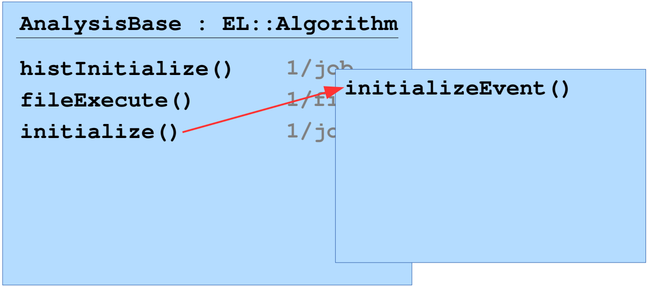
CxAODMaker workflow

```
AnalysisBase : EL::Algorithm

histInitialize()     1/job
fileExecute()        1/file
initialize()         1/job
```

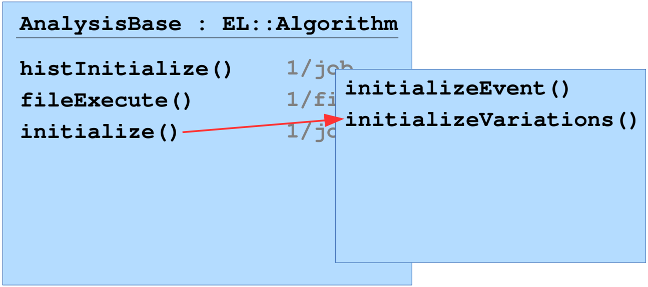
AnalysisBase

- initialize() is the main method for initializing the analysis
- it is split into several sub-methods to factorize the code (and easier derivation)...



AnalysisBase

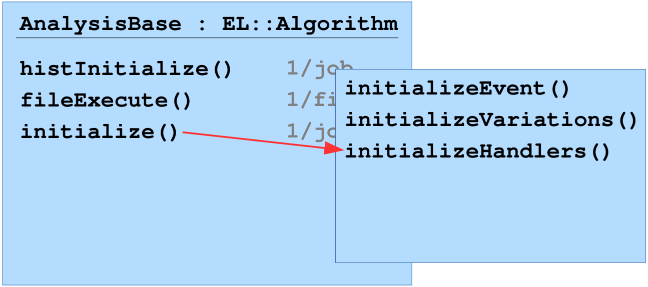
- initializeEvent() reads the configuration, prints sample info and books the output file



AnalysisBase

- initializeEvent() reads the configuration, prints sample info and books the output file
- initializeVariations() reads the desired systematic variations from the config file

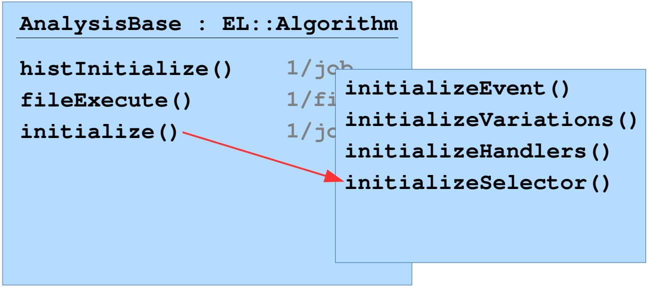
CxAODMaker workflow



AnalysisBase

- initializeEvent() reads the configuration, prints sample info and books the output file
- initializeVariations() reads the desired systematic variations from the config file
- initializeHandler() creates the object handlers: one for each collection of el, mu, jet...

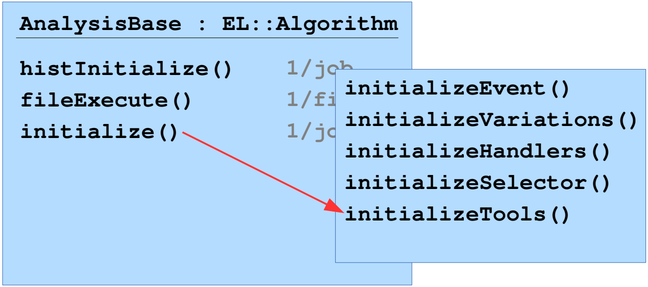
CxAODMaker workflow



AnalysisBase

- initializeEvent() reads the configuration, prints sample info and books the output file
- initializeVariations() reads the desired systematic variations from the config file
- initializeHandler() creates the object handlers: one for each collection of el, mu, jet...
- initializeSelector() books the "EventSelector", taking care of event selection and OR

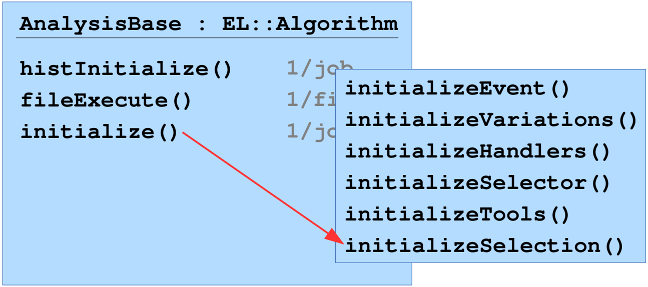
CxAODMaker workflow



AnalysisBase

- initializeEvent() reads the configuration, prints sample info and books the output file
- initializeVariations() reads the desired systematic variations from the config file
- initializeHandler() creates the object handlers: one for each collection of el, mu, jet...
- initializeSelector() books the "EventSelector", taking care of event selection and OR
- initializeTools() asks the object handlers to initialize their CP tools

CxAODMaker workflow



AnalysisBase

- initializeEvent() reads the configuration, prints sample info and books the output file
- initializeVariations() reads the desired systematic variations from the config file
- initializeHandler() creates the object handlers: one for each collection of el, mu, jet...
- initializeSelector() books the “EventSelector”, taking care of event selection and OR
- initializeTools() asks the object handlers to initialize their CP tools
- initializeSelection() determines which event selection should be run from the config file

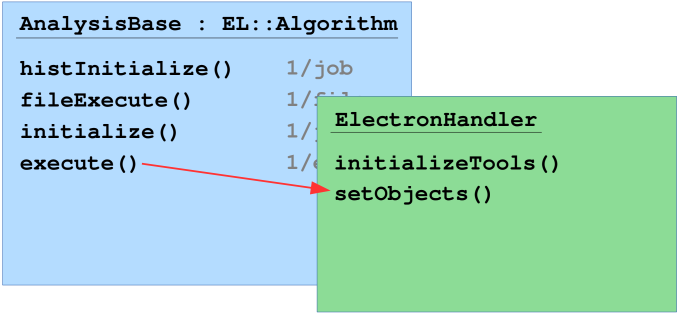
CxAODMaker workflow

AnalysisBase : EL::Algorithm	
histInitialize()	1/job
fileExecute()	1/file
initialize()	1/job
execute()	1/event

AnalysisBase

- execute() is the main method that is called for each event
- in our design this method is quite compact

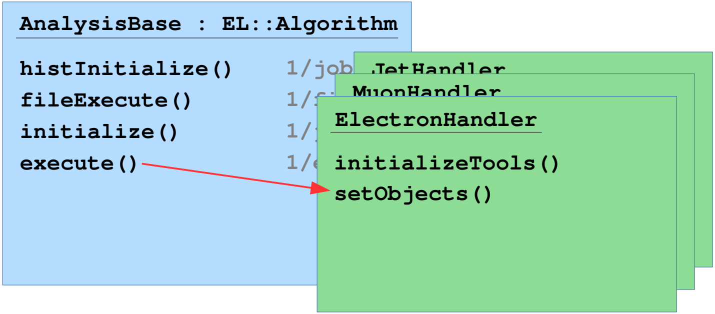
⇒ the heavy work is delegated to other classes...



ObjectHandler

- all calibration happens inside the object handlers, e.g. ElectronHandler
- the CP tools are created in initializeTools()
- setObjects() creates shallow copies of the input container
- one for each variation affecting this object type (only electron systematics for electrons)

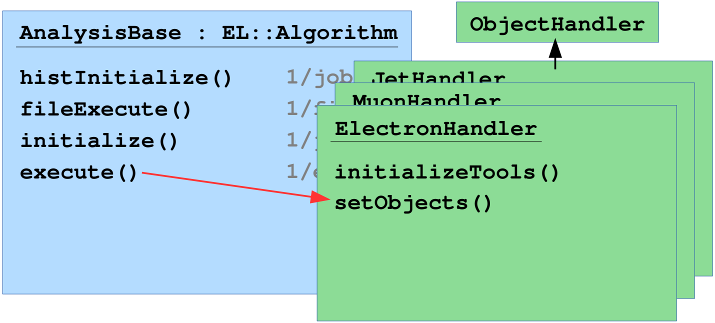
CxAODMaker workflow



ObjectHandler

- all calibration happens inside the object handlers, e.g. ElectronHandler
- the CP tools are created in initializeTools()
- setObjects() creates shallow copies of the input container
- one for each variation affecting this object type (only electron systematics for electrons)
- AnalysisBase::execute() loops over all object handlers and calls their setObjects()

CxAODMaker workflow

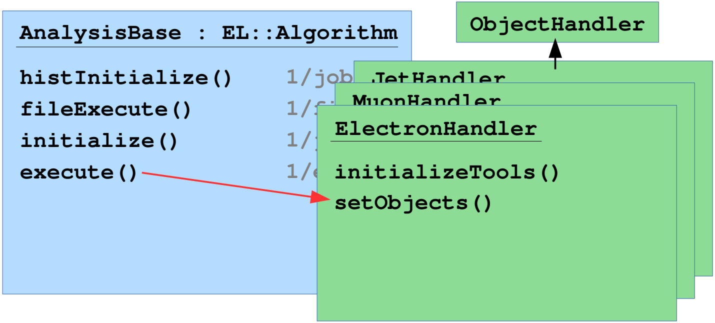


ObjectHandler

- note that you won't find a setObjects() method in the ElectronHandler
- this method is the same for all object types, so it is defined in the ObjectHandler class

⇒ reduce duplication of code and avoid inconsistencies

CxAODMaker workflow



ObjectHandler

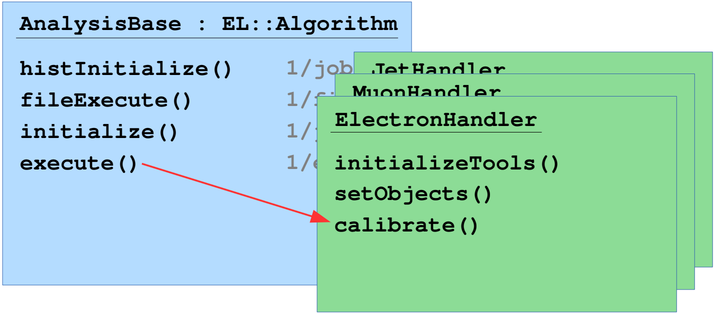
- note that you won't find a setObjects() method in the ElectronHandler
- this method is the same for all object types, so it is defined in the ObjectHandler class

⇒ reduce duplication of code and avoid inconsistencies

- ObjectHandler is a templated class: cannot be put into a vector (and is a bit hard to read)

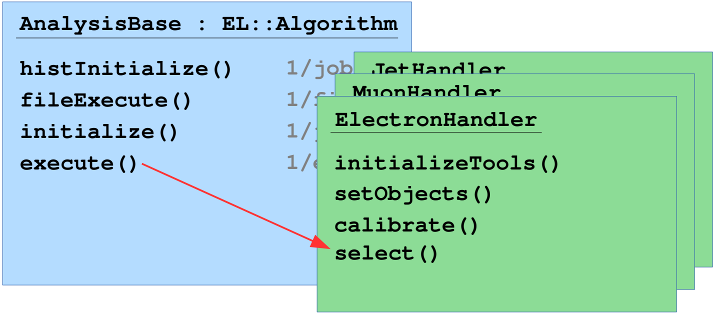
⇒ there is another purely virtual ObjectHandlerBase, so we can fill a vector and do loops

CxAODMaker workflow



ObjectHandler

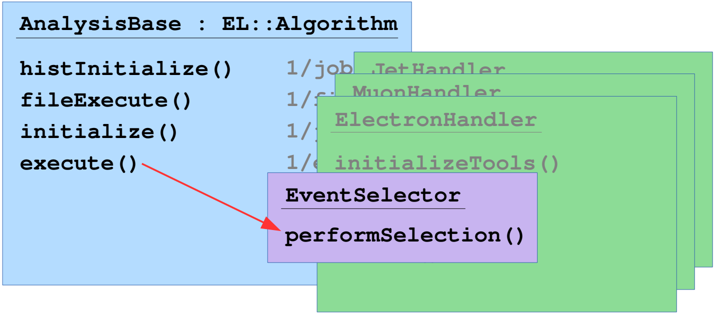
- calibrate() calls the CP tools and applies the calibration to the shallow copies
- this is done for each object and systematic variation
- some information is decorated to the objects, e.g. a isVeryLooseLH flag for electrons



ObjectHandler

- select() calls a number of "pass" methods to select objects, e.g. passVHLooseElectron()
- properties of the objects are tested and the result is decorated, e.g. isVHLooseElectron
- special flag: passPreSel determines whether the object is used in the overlap removal

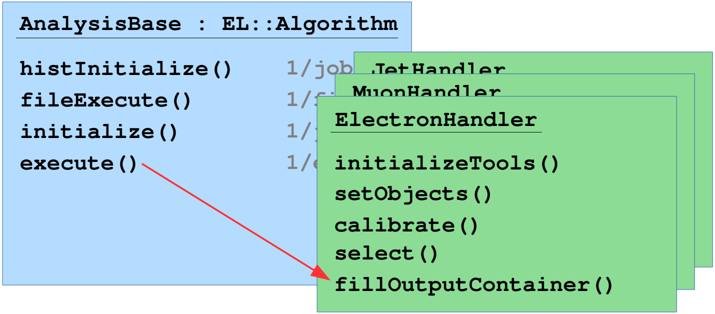
CxAODMaker workflow



EventSelector

- the EventSelector is called after object selection (not to be confused with EventSelection)
- performSelection() implements a loop over all systematic variations:
  1) the overlap removal is done for each variation
  2) the event selection is called directly afterwards
- the event passes if at least one variation passes the selection
- the overlap removal writes a passORGlob to the objects:
  it is true if the object passed the OR in at least on variation

CxAODMaker workflow



ObjectHandler

- fillOutputContainer() is called only if the event passes the selection
- the output containers are created and input objects are copied over:
- only objects that passed the selection, determined by a checkPassSel() method, are written
- the information that is written for each object is determined in setVariables()

CxAODMaker workflow

```
AnalysisBase : EL::Algorithm

histInitialize()    1/job
fileExecute()       1/file
initialize()        1/job
execute()           1/event
```

AnalysisBase

- the METHandler and EventInfoHandler are not derived from ObjectHandler

⇒ their structure is a bit different, calls from execute() are separated

- if the event did not pass the selection wk()->skipEvent() is called

⇒ prevents that any other EL algorithm (i.e. the TupleMaker) is called after AnalysisBase for this event

CxAODMaker workflow

AnalysisBase : EL::Algorithm	
histInitialize()	1/job
fileExecute()	1/file
initialize()	1/job
execute()	1/event
postExecute()	1/sel.evt

AnalysisBase

- postExecute() is called only if the event has passed the selection, writes it to the output

CxAODMaker workflow

AnalysisBase : EL::Algorithm	
histInitialize()	1/job
fileExecute()	1/file
initialize()	1/job
execute()	1/event
postExecute()	1/sel.evt
finalize()	1/job

AnalysisBase

- postExecute() is called only if the event has passed the selection, writes it to the output
- finalize() closes the output file and prints some info

CxAODMaker workflow

AnalysisBase : EL::Algorithm	
histInitialize()	1/job
fileExecute()	1/file
initialize()	1/job
execute()	1/event
postExecute()	1/sel.evt
finalize()	1/job
histFinalize()	1/job

AnalysisBase

- postExecute() is called only if the event has passed the selection, writes it to the output
- finalize() closes the output file and prints some info
- histFinalize() writes the "MetaData_EventCount" histogram

Possibilities for object decoration

1) Directly via auxdata / auxdeco

- Very easy, it's a one-liner
- However, this is very slow (due to string hashing), wouldn't recommend to use it

```
electron->auxdata< float >("ptcone20") = 0.1; // for non-const objects
electron->auxdecor< float >("ptcone20") = 0.1; // for const objects
```

2) Using an Accessor or Decorator

- This is very fast (if the accessor/decorator is initialized only once)
- However, the declaration of the decorations can be distributed all over the code

```
static SG::AuxElement::Accessor< float >  acce_ptcone20("ptcone20"); acce_ptcone20(*electron) = 0.1;
static SG::AuxElement::Decorator< float > deco_ptcone20("ptcone20"); deco_ptcone20(*electron) = 0.1;
```

3) Using a build in method of the object

- This should be fast (depends in the implementation)
- Not clear how it behaves for const / non-const objects

```
electron->setIsolationValue(0.1, xAOD::Iso::ptcone20);
```

4) Using our ObjectDecorator

- Uses Accessors / Decorators internally depending on const / non-const
- Is very fast and the definition of variables are gathered at one place

```
m_decorator.set(electron, ElecFloatProps::ptcone20, 0.1);
```

Would recommend to use method 3 if available and 4 otherwise!

Session 2

- Adding a new variable
- Grid running
- Run CxAODReader

Details on CxAODReader

CxAODReader layout

CxAODReader

- the main class is AnalysisReader, an EventLoop algorithm (similar to AnalysisBase in CxAODMaker)
- however, the structure is much simpler
- histInitialize() books the desired histograms
- initialize() sets the event selection, read sum of weights and cross sections
- execute() loops over all systematic variations and calls “fill” methods

Implication of shallow copies

- CxAODMaker stores an event / object if any variation passed

⇒ one has to re-run overlap removal, object and event selection in the reader

Updates in the trunk

- the CxAODReader has undergone some structural changes
- ObjectReader takes care of discovering and reading all variations for a given container
- HistNameSvc organizes histograms by name, e.g.

  ```
  SysMUONS_SCALE__1up/Zcc_pretag3jet_vpt0_pTB2_SysMUONS_SCALE__1up
  ```
- HistSvc makes booking and filling a histogram a one-liner:

  ```
  m_histSvc -> BookFillHist("pTB2", 100, 0, 500, bdt_pTB2/1e3, m_weight);
  ```

Session 3

- Adding a new histogram
- Making plots

Implementing a new analysis

Strategy for new analyses

Strategy for new analyses

- The "base" code implements the VHbb analysis
- New functionality (e.g. a different object selection) is added by deriving new classes
- These classes can be put into new packages to separate the analysis

Example: modifying the muon selection

- Derive a new MuonHandler_Example from MuonHandler
- The new class inherits the functionality from the base class and can extend it, e.g. by a new "pass" method
- A new AnalysisBase_Example is needed to use the new handler
- Finally, a new executable is needed

Session 4

- Creating new packages
- Adding new classes
- Adding a new executable

Backup

Performance of CxAODMaker

Sample specifics

- HIGG2D4 DxAOD (2 lepton filter)
- 10000 ttbar events (2.3 Gb)
  (mc14_8TeV.117050.PowhegPythia_P2011C_ttbar.merge.DAOD_HIGG2D4.e1727_s1933_s1911_r5591_r5625_p1784)

Reading/writing

- ElectronCollection
- Muons
- AntiKt4LCTopoJets
- CamKt12LCTopoJets
- AntiKt4ZTrackJets
- MET_RefFinal

Configuration

- 10 systematic variations
- VHbb "2 lepton" event selection

Processing time (CPU time)

- about 120 evt/sec

Disk space

- Output CxAOD file size 3.5 Mb (reduction factor: 650)