AnalysisTopDocumentation

Not yet Certified as ATLAS Documentation **Responsible:** TomasDado

Introduction

This TWiki provides a short description of the AnalysisTop packages, their purpose and structure. (These packages have been merged into AnalysisBase project since 11.2019)

For beginners who want to have a quick hands on how to use AnalysisTop as a black box, you can go to: TopxAODStartGuideR21. But you are still recommended to read this twiki to get a deeper idea of how AnalysisTop works internally, which will in turn helps you digest the above start guide better.

For TopReco liaisons, contacts and managers, the twiki helps you to quickly locate the package relevant for your mandate. You are also welcome to contribute to the construction of the twiki.

The AnalysisTop packges live in the cern git: Athena of branch 21.2. Please submit merge request if you want to make any update: AnalysisTopGuideToUpdatingAthena

List of AnalysisTop packages

The following packages are part of AnalysisTop:

Package name	Link to gitlab	Short description	
TopDataPreparation	athena link	Code to read TopDataPreparation files with xSection values and shower gerators	Marco
TopAnalysis	athena link	Contains base object and event selection, also contains `top-xaod` executable	Marco
TopCPTools	athena link	Contains the initialization of all CP tools	Oliver
TopConfiguration	athena link	Code that read the configuration file and sets the appropriate variables	Done
TopCorrections	athena link	Code that applies corrections to objects, e.g. scale factors	Marco
TopEvent	athena link	Code that controls all variables that are read event by event	Marco
TopEventReconstructionTools	athena link	Code for ttbar kinematics reconstruction, e.g. KLFitter	Tomas
TopEventSelectionTools	athena link	Definition of the basic selection options available	Tomas
TopExamples	athena link	Contains macros and small codes to compare outputs, submit to grid scripts etc.	
TopHLUpgrade	athena link	Contains code needed for HL LHC upgrade	Yichen
TopJetSubstructure	athena link	Self explanatory	
TopObjectSelectionTools	athena link	Contains basic object selection code	Oliver
TopParticleLevel	athena link	Code for particle level processing	Yichen
TopPartons	athena link	Code for parton level analysis and identification of objects from top decays	Yichen
TopSystematicObjectMaker	athena link	Code where different CP tools are applied to objects	Marco

General description of the top-xaod executable

The top-xaod is located in TopAnalysis package. It steers the whole activity of producing flat ntuples out of xAOD and DAOD/TRUTH derivations. Usually users don't have to touch it, since it uses interfaces provided from all other packages and it's usually the implementation of those interfaces that are constantly updated. But it's good to have a understanding of the work flow of top-xaod, as it follows the general steps of data analysis.

Firstly, it checks for metadata of the input file, e.g. whether it is MC or data and whether it is AOD, TRUTH, or DAOD. These metadata will be stored in an instance of the TopConfig class. These information are needed in later step or other packages to make decisions.

Then tools (either ASG tools or local tools) are initialized in order (important, since local tools may require the initialization of ASG tools first), e.g. ASG CP tools, EventCleaningSelection, ObjectMaker, EventSelectionManager, ParticleLevel and Upgrade Loader, TopPartonHistory algorithm, LHAPDF reweighting calculator, EventMaker, ScaleFactor calculater, MMWeightIFF Tools, and lastly EventSaver.

Before finally starting the event loop, sumWeight trees are defined to collect sum of weights (nominal or sys variation).

In event loop, the particle level (or upgrade) event is made and selected. If it's TruthDAOD, then the code stops. Otherwise, it enters reco level selection. GRL, PV, Trigger, BadCalo selections are applied firstly. Only for events surviving the above selections that the ObjectMaker, ObjectSelection, ScaleFactorCalculater are called, including nominal and systematic variations. After that, tight and loose events are made by EventMaker, and event selection defined by the cut file is applied to these events. Events passing the selection are saved via the EventSaver, which is the class that defines the branches in the final flat ntuple.

Individual package description

TopDataPreparation

This is an helper package that can be used to read XSection files more easily. The top-xaod script in particular uses a SampleXsection object, defined in this package, to read info about the sample from dedicated txt files. These files are not stored anymore in the package, but they are in a dedicated cvmfs area: /cvmfs/atlas.cern.ch/repo/sw/database/GroupData/dev/AnalysisTop/TopDataPreparation/

The file interesting for most analyzers is XSection-MC16-13TeV.data

This has lines like

```
410501   397.11    1.1390   pythia8
```

where the first column is the dsid, the second one is the xsec*filter_efficiency, the third one is the k-factor, the fourth one is the showering algorithm. The showering algorithm is needed to setup correctly the b-tagging SF tools. More info are available here: https://twiki.cern.ch/twiki/bin/view/AtlasProtected/TopxAODStartGuideR21#B_tagging_showering_algorithm_an

TopAnalysis

In this package classes controlling the output produced by AnalysisTop are defined, as well as the top-xaod exectuable (the latter is described above).

EventSaverFlatNtuple

Inheriting from EventSaverBase, this class defines the default event saver used by AnalysisTop. It takes care of defining the branches for the output trees and to fill them appropriately by retrieving the info from other AT classes. Several trees are declared (using the TreeManager helper class) and filled here:

- the "nominal" output tree and the trees corresponding to systematic variations, which are the ones tipically used by the users;
- the "xxx_Loose" trees which can be used e.g. for multijet background estimation;
- the "truth" tree, used to store some truth-level information for each event: few variables are always saved by default (event number, mu, mc_weight...), and then some additional output that can be stored by using the TruthBlockInfo (beware: this will cause saving all truth particles basic info and therefore dramatic disk size consumption) and the PDFInfo options, see https://twiki.cern.ch/twiki/bin/view/AtlasProtected/TopxAODStartGuideR21#Truth_options
- the "particleLevel" tree, which is a truth-level replica of the reco-level tree, storing truth-level objects passing the particle level selection, see https://twiki.cern.ch/twiki/bin/view/AtlasProtected/TopParticleLevel
- the "upgrade" tree, used for upgrade studies

If you want to add/remove some variables in output for everyone in AnalysisTop, you have to do it here. But please check with AT managers and top-reco conveners before doing that, variables containing duplicated information or that are anyway interesting only for a specific analysis shoulnd't be included here but in a dedicated custom event saver.

MetadataTree

This class handles the creation and filling of the sumWeightsTree tree, used for the normalization of the samples, and the sumPdfWeights trees, useful when using LHAPDF for PDF studies.

TopCPTools

The TopConfiguration package is the central package for handling of CP tools used within AnalysisTop. Its primary purpose is to initialize CP tools in accordance with configuration options provided by users via `TopConfig` class. In general, all CP tools used, regardless of where in AnalysisTop they are used, should be configured and initialized in this package. The individual CP tools are initialized in subclasses based on the type of object they act upon, e.g. `TopEgammaCPTools` manage CP tools for electrons and photons, for example the `EgammaCalibrationAndSMearingTool`.

We will focus here on documenting the two most common operations an AT developer might perform.

1. Changing settings of CP tools -- most commonly due to new recommendations or extended features of CP tools. This mostly involves modifying existing instances of CP tools within the TopCPTools package. 2. Adding new CP tools -- this involves making sure dependencies in `CMakeLists.txt` of the TopCPTools package reflect any addition of new package, adding the new CP tool ToolHandle in respective class in TopCPTools and appropriate configuration of the CP tool.

Below we will discuss the general idea how CP tools are initialized, "book-kept" and retrieved elsewhere in AnalysisTop in order to execute them on xAOD objects. This is the minimum necessary knowledge in order to understand how to add new CP tool for use within AnalysisTop.

Initializing CP tools and passing ownership to ToolStore

These classes gathering CP tools inherit from `asg::AsgTool`. Typically, the individual CP tools within these classes will be handeled via ToolHandles, for example:

```
class JetMETCPTools: public asg::AsgTool {
public:
   explicit JetMETCPTools(const std::string& name);
   virtual ~JetMETCPTools() {}

   StatusCode initialize(); // this is where the individual CP tools will be initialized and configured

private:
   // each tool should normally be a ToolHandle
   ToolHandle<ICPJetUncertaintiesTool> m_jetUncertaintiesTool;
};
```

This means that once the tool is created it will exist in the ToolStore under a unique name and can be retrieved from the ToolStore elsewhere in the AnalysisTop codebase without having to pass pointer or reference to access the tool -- all one needs is the unique name associated with the ToolHandle. One can dynamically allocate memory for the CP tool, keep the address in a pointer and then pass this pointer to the ToolHandle, where the ToolHandle will take control over the ownership of the tool.

```
// some code ....

ICPJetUncertaintiesTool* tool = new JetUncertaintiesTool("name of the tool");
// configure the various properties of the tool
top::check(tool->setProperty("SomeProperty", some_value), "Print this message if setting property fails");
m_jetUncertaintiesTool = tool; // pass the pointer to the ToolHandle which will take ownership
```

Retrieving CP tools from ToolStore elsewhere in code

Above we've shown an example how to initialize a CP tool and make sure it is registered in the ToolStore via the ToolHandles. One can then define a ToolHandle to the CP tool elsewhere in the code and retrieve the tool:

```
// some code ....

ToolHandle<ICPJetUncertaintiesTool> tool("name of the tool");
top::check(tool.retrieve(), "Print this message if retrieving the tool fails");
tool->doSomething(some_xAOD_object); // methods and members of the tool are accessed as in a pointer
```

Notes on implementing new CP tool instances

As already mentioned, there are (not-so-frequent) cases when a completely new CP tool needs to be added in order to extend functionality in AnalysisTop. A good example of this is the recent addition of scale factors for calibrating boosted top/W/Z taggers for large-radius jets. This involved addition of completely new instances of CP tools. One has to make sure that in case any new package is needed to be included to use the new CP tool, the `CMakeLists.txt` in the TopCPTools package is updated. The following lists in the `CMakeLists.txt` should be updated: `atlas_depends_on_subdirs` with the name of the new package that TopCPTools will require to function (package dependency) and `atlas_add_library` with the name of the corresponding library built from this package dependency.

TopConfiguration

The TopConfiguration package mostly takes care of managing the options set by the user in the config file when running the top-xaod executable.

ConfigurationSettings

Implementation of the ConfigurationSettings class. This is the class which takes care of declaring the parameters that can be set by the user in the config file provided to top-xaod.

In most cases developers will not need to modify the header. In the cxx file, in the class constructor the registration of all parameters that can be used in a config file is done, with lines like:

```
registerParameter("ParameterName", "Description of the parameter printed when running top-xaod; describe all possible options for the parameter! ", "DefaultValueOf
```

So when a developer needs to add a new parameter configurable by the user in the config-file, a new "registerParameter" line will have to be added.

TopConfig

Implementation of TopConfig class. This is the class which is effectively storing the configurations used by the top-xaod program, e.g. this is where we store which collections of objects the users want to run on, the pt/eta/etc.. cuts, and so on. When a developer wants to add a new configurable parameter for AnalysisTop, a new variable (with appropriate set/get methods) must be added in the header in the appropriate places (look at the header to see many examples on how this is done).

Furthermore, in the TopConfig::setConfigSettings method the reading of a parameter (declared in ConfigurationSettings) is performed. If the developer added a new configurable parameter, a line like this needs to be added:

```
this->parameterSetMethod(settings->value("ParameterName"));
```

This will read the settings set in the config file and will store them in the TopConfig object which is used to steer the execution of top-xaod. It's always good practice to protect this kind of readings with a try and catch structure, see an example here.

TopCorrections

In this package several classes are defined to manage the calculation and retrieval of Scale Factors. The retrieval in particular is handled by the ScaleFactorRetriever class, which has both methods to retrieve the per-object SFs and methods dedicated to calculate from them the per-event global SF (e.g. the lepton SF which combines all kind of SFs for all leptons in one event). In this class also all available systematic variations for SFs in AT are declared.

For the calculation, there are dedicated classes for each object type, e.g. MuonScaleFactorCalculator. These classes take care of retrieving the official CP tools (which were declared and configured in the TopCPTools package), applying appropriate systematic variations and use them to decorate physics objects with the scale factors (nominal and systematic variations).

TopEvent

This package defines the classes needed to contain the event-level information.

TopEvent

This is the main class to manage event-lvel information in AnalysisTop. It has publicly accessible containers holding objects passing the selection defined by the user in the config file (e.g. m_electrons, m_primaryVertices, ...), event information (accessible via the const xAOD::EventInfo* m_info public memebr variable) and other information (e.g. it stores the KLFitter results, accessible via the m_KLFitterResults pointer). For example, custom event saver have saveEvents methods getting Event objects in input, and can use those to access selected objects

```
void CusomSaver::saveEvent(const top::Event& event)
{
  [...]

  for(xAOD::Electron* el : event.m_electrons)
 {
   //do domething on electrons
 }

  [...]
}
```

SystematicEvent

This is an helper class, holding list of indices of objects passing the event selection, and with indices used to identify the specific systematic variation for which it is used and the corresponding output tree, used to then create TopEvent objects.

EventTools

In this class some useful helper methods are defined. The most notable one is **top::check**, which can be used when methods needing a check on a StatusCode are called, e.g.

```
top::check(evtStore()->retrieve(m_truthParticles, m_topconfig->sgKeyMCParticle()), "Failed to retrieve TruthParticle Container");
```

So don't forget to include this class if you need to use top::check.

TopEventMaker

This is the class responsible of actually creating the Event objects. It reads information from the xAOD event, it takes care of retrieving object containers, and for each systematic variation preare a corresponding TopEvent object holding all relevant physics objects.

TopEventReconstructionTools

The package is contains several algorithms to reconstruct kinematics of top quarks from their decay products in ttbar(+X) topologies. The algorithms have several levels of implementation, from fully integrated to AnalysisTop like KLFitter to only partial implementation (true for most algorithms).

Chi2LJets

Provide a simple chi2 reconstruction in lepton+jets channel. Is not fully implemented.

KLFitterRun

Main code that runs KLFitter. KLFItter is *not* part of AnalysisTop nor AnalysisBase, it is implemented via AtlasExternals as an external package that is only linked.

KLFitterTool

Code that steers KLFItter. Sets all parameters like type of likelihood, fixed/floating top mass etc. Method `execute` passes all required information to KLFitter (jets, leptons etc) and actually runs the algorithm and stores the output in a special container.

MT2Reco

MT2 algorithm has only a dummy implementation that does nothing.

NeutrinoWeighting

Code that runs Neutrino Weighting (algorithm for dilepton ttbar filan state). Is not implemented in AnalysisTop, but the package can be included in a custom event saver.

PTMaxReco

One of the simplest reconstruction algorithms, not implemented in AnalysisTop, but can be included i na custom event saver.

PseudoTopReco

Code that steers the PseudoTop algorithm, is fully implemented in AnalysisTop.

PseudoTopRecoRun

Code that executes PseudoTop algoritm.

Sonnenschein

An algorithm to reconstruct ttbar dilepton events using exact solution of the equations (4th order polynomial). Is not fully implemented in AnalysisTop.

SonnenscheinEngine

The internals of the reconstruction algorithm.

TtresChi2

Yet another implementation of chi2 algorithm, used by ttbar resonance group. Not implemented in AnalysisTop.

TtresNeutrinoBuilder

Code to calculate neutrino four momentum in lepton+jets final state. Not implemented in AnalysisTop.

TtresdRmin

Code to calculate top mass using the delta R method. Not implemented in AnalysisTop.

TopEventSelectionTools

This package contains the different selection options that are supported by AnalysisTop. Note that not all option actually apply selection, some of the simply call some tools. All selectors also appear in a cutflow in the same order as they appear in the config file. The table below summarizes the individual source codes.

Source file	Config option	Short description	Comments
ExamplePlots	EXAMPLEPLOTS	Produces some basic histograms	
FakesMMConfigs	FAKESMMCONFIGS	Applies the selected MM configs for the event	
GRLSelector	GRL	Applies election based on the Good Run List	
GlobalTrigDecisionSelector	GTRIGDEC	Applies trigger decision selection	Respects the trigger thresholds
GlobalTrigMatchSelector	GTRIGMATCH	Applies trigger matching selection	
GoodCaloSelector	GOODCALE	Selects only events with good calorimeter conditions	
HTSelector	HT	Selects events based on HT (sum of all electrons, muons and jets)	Works also for particle level
InitialSelector	INITIAL	A dummy selector, useful for cutflows	
JetCleaningSelector	JETCLEAN	Applies jet cleaning	Also applies BadBatman cleaning when selected
JetFlavorPlots	JETFLAVORPLOTS	Not a selector. Saves histograms with information	

		needed for custom quark-gluon composition	
JetFtagEffPlots	JETFTAGEFFPLOTS	Not a selector. Saves histograms with f-tag efficiencies	
JetNGhostSelector	JET_N_GHOST (X)	Ghost matching selection	Works only on particle level
KLFitterSelector	KLFitter	Not a selector. Runs KLFitter algorithm on a given event	
METMWTSelector	MET+MWT	MET+MWT selection	Exits code if no electrons or muons are present. Uses the first electron if present, if not uses the first muon. Works on particle level
METSelector	MET	Simple MET selector	Works on particle level
MLLSelector	MLL	Selects events based on invariant mass of 2 leptons	Requires exactly 2 leptons, otherwise crashes. Works on particle level
MLLWindowSelector	MLLWIN	Selects events outside of 2 lepton invariant mass	Requires exactly 2 leptons, otherwise crashes. Works on particle level
MV2c10Selector	MV2C10_N	Applies btag selection	
MWTSelector	MWT	Aplies MWT selection	Exits code if no electrons or muons are present. Uses the first electron if present, if not uses the first muon. Works on particle level
NElectronNMuonSelector	EL_N_OR_MU_N	Selects N electrons + muons (combined)	Works on particle level
NElectronNMuonTightSelector	EL_N_OR_MU_N_TIGHT	Same as above, but also requires tight selection	
NElectronSelector	EL_N	Selects N electrons	
NElectronTightSelector	EL_N_TIGHT	Same as above, but also requires tight selection	
NFwdElectronSelector	FWDEL_N	Selects N forward electrons	
NFwdElectronTightSelector	FWDEL_N_TIGHT	Same as above, but also requires tight selection	
NJetBtagSelector	JET_N_BTAG	Selects N b-tagged jets	Works with TJET_N_BTAG for track jets
NJetSelector	JET_N	Select N jets	Works on particle level
NLargeJetSelector	LJET_N	Selects N large jets	Works on particle level
NMuonSelector	MU_N	Selects N muons	Works on particle level
NMuonTightSelector	MU_N_TIGHT	Same as above, but requires tight muons	Works on particle level
NPhotonSelector	PH_N	Selects N photons	Works on particle level
NRCJetSelector	RCJET_N	Selects N reclustered jets	Works on particle level
NSoftMuonSelector	SOFTMU_N	Select N soft muons	Returns true on particle level
NTauSelector	TAU_N	Selects N taus	Works on particle level
NTauTightSelector	TAU_N_TIGHT	Same as above, but requires tight taus	Works on particle level
NVarRCJetSelector	VRCJET_N	Selects N varaible R reclustered jets	Works on particle level
NoBadMuonSelector	NOBADMUON	Removes bad muons	
OSLeptonSelector	OS	Selects events with opposite sign of leptons	Selects events with at least one positive and at least one negative lepton. Works on particle level.
OSLeptonTightSelector	OS_TIGHT	Same as above, but require tight leptons. Works on particle level.	
ParticleLevelSelector	PARTICLE_LEVEL	Disable reco level, enable particle level for selectors	
PlotManager	IS NOT A SELECTOR	A helper class for producing plots	
PrimaryVertexSelector	PRIVTX	Selects events with at least one primary vertex	
PrintEventSelector	PRINT	Prints cutflow	
RecoLevelSelector	RECO_LEVEL	Enable reco level, disable particle level for selectors	
RunNumberSelector	RUN_NUMBER	Selects events based on their run number	Uses RandomRunNumber for MC
SSLeptonSelector	SS	Selects events with the same sign for leptons	Selects events with at least 2 positive or 2 negative leptons. Works on particle level.
SSLeptonTightSelector	SS_TIGHT	Same as above, but requires tight leptons. Works on particle level.	
SaveEventSelector	SAVE	Crucial setting that tells the code to save the output to a root file	
SignValueSelector	IS NOT A SELECTOR	A helper code to parse other options	
TrackJetCleaningSelector	TRACKJETCLEAN	Applies cleaning to track jets	
TreeManager	IS NOT A SELECTOR	A helper code to manage trees	
TrigDecisionLooseSelector	TRIGDEC_LOOSE	Applies (deprecated) trigger decision for loose leptons	
TrigDecisionSelector	TRIGDEC	Same as above but for tight leptons	
TrigDecisionTightSelector	TRIGDEC_TIGHT	Same as above but for tight leptons	
TrigMatchSelector	TRIGMATCH	Applies (deprecated) trigger matching, also saves the	When global triggers are use the trigger decision is also

		trigger decision	stored

TopExamples

TopHLUpgrade

This package is used to study physics potential at the HL-LHC. Truth events (TruthParticles and TruthJets) are the input. Then the package tries to mimic what a reconstruction level event will look like at the HL-LHC by: smearing the truth particles and jets and adding pile-up effects (jets), via using parametrized functions instead of real simulation. Fake objects (lepton and photon) are also simulated using parametrized fake rates. The event after the above procedure is ready to be selected. The selection shares the same infrastructure as the particle level event selection defined in the TopParticleLevel package, which means the cuts are defined with a common string in the cut file and the same selection functions are used.

The key of this package is the class that converts the truth event into a pseudo reco-level event, which is the UpgradeObjectLoader.cxx. The actual parameterization functions for the HL-LHC condition are defined in this class UpgradePerformanceFunctionsxAOD.cxx.

TopJetSubstructure

TopObjectSelectionTools

TopParticleLevel

This package is used to do particle level study. By particle level, it usually means particles with mean lifetime > 0.3×10^{-10}s.

Truth events (TruthParticles and TruthJets) are the input, the names of which can be specified in the cut file. e.g. you have the freedom to use different truth jet collections predefined in the truth derivation. Before performing truth object selection, truth leptons are dressed with photons within a cone of dR < 0.4 around it. Truth photons used for dressing are removed from the candidate photon list. Then objection selections are performed via different selectors in the package TopParticleLevel. The selected objects undergo overlap removal, which is similar but different to the reco-level object overlap removal. The particle-level event level selection is defined in the TopEventSelectionTools package. Selectors defined in this package can be dual use: it can have two implementations, one for reco level and the other for particle level.

The class that steers the particle-level object selection is the ParticleLevelLoader.cxx.

To know the exact triggering of the particle level algorithm, check the ``m_active`` variable in ParticleLevelLoader.cxx.

A much more detailed twiki of this package is available TopParticleLevel.

TopPartons

The parton history package is used to retieve the information (e.g. 4-momentum and ID) of hard partons (e.g. top and W) as well as their decay products. The information can be extracted at different stage of the event generation: before FSR, after FSR, after FSR but before decay (if unstable parton).

For hard processes with different topologies, the algorithms used to sort out the above information are different. In TopPartonLevel package, algorithms for the following hard processes are available: ttbar, ttbar+Z, ttbar+gamma, ttbar+light parton, tHq, tbbar, and Wtb. You are welcome to add parton level algorithm to the repository for your own signal topology.

For now, the initialization of a parton algorithm is done in a hard-coded way in top-xaod.cxx, which should be improved in the future. It means, to have a new parton algorithm, you have to also update the top-xaod.cxx.

TopSystematicObjectMaker

This is the package that prepares the collections of physics objects, by decorating them with variables which are then needed in other parts of the AT software. The ObjectionCollectionMaker class takes care of steering the execution of all the other classes of this package, which are dedicated to specific objects (e.g. MuonObjectCollectionMaker). In the specific maker things like isolation variables, other additional variables (e.g. d0 significance), are added; momentum calibration of the physics objects is also performed.

Things to check when doing a merge request

Always try to follow the ATLAS c++ style guides. The code has been written and maintained by many people so it should be as readable as possible. Even small things like wrong indetation can make the code very hard to read. Try to always use `git diff` before you commit a change to review your updates. If you made many changes and the indentation is wrong, you can try to use `uncrustify` to fix the indentation in a whole file link.

Useful information

Some basic concepts of the software infrastructure will be summarized here. These are general concepts in athena and not only ties to AnalysisTop code

Object decorations

So called "decorations" are used to pass information between different parts of the code. Decorations are variables attached to different `xAOD::` objects, e.g. `xAOD::Particle`. This allows to add some information, e.g. to electrons in one part of the code and read the information in a different part of the code.

There are two ways how to decorate and object:

```
// One way how to decorate an object
for (xAOD::Jet* jet : jets) {
  jet->auxdecor<float>("variable") = 1;
}
```

The code above added decoration with name "variable" to each `xAOD::Particle*` object in a vector of jets, and set the value to 1.

The second option how to decorate an object is:

```
// Other way how to decorate an object
static SG::AuxElement::Decorator<float> variable("variable");
for (xAOD::Jet* jet : jets) {
  variable(*jet) = 1;
}
```

Which does exactly the same as the version before. There are differences between these two options that basically impact only performance, where the latter option is the preferred one as it has slightly better performance.

To read the decoration, you can do:

```
xAOD::Jet* jet;
/*

// some code //

*/

// check if the decoration exists
if (jet->isAvailable<float>("variable")) {
  const float variable = jet->auxdataConst<float>("variable");
}
```

If you access a decoration that does not exist, the code will throw an exception, e.g. `exception: SG::ExcBadAuxVar: Attempt to retrieve nonexistent aux data item `::mcEventWeights' (508).`

Similary to SG::AuxElement::Decorator for decorating objects, for retrieving aux data, `SG::AuxElement::ConstAccessor` can be used as opposed to using `auxdataConst` method. The `ConstAccessor` approach has slightly better performance.

```
static const SG::AuxElement::ConstAccessor<float> acc("variable");
xAOD::Jet* jet;
/*

// some code //

*/

// check if the decoration exists
if (acc.isAvailable(*jet)) {
  const float variable = acc(*jet);
}
```

Shallow copies

Major updates:
-- TomasDado - 2019-11-18

Responsible: TomasDado
Last reviewed by: **Never reviewed**

Topic revision: r22 - 2020-02-12 - YichenLi