Observables

April 1, 2015

Carsten Burgard

Observables

tree

$\vdots$

$x_1\ x_2\ x_3\ x_4\ \ldots$

$\vdots$

histogram input

cut value

event weight

...

Introduction

Booking Histograms

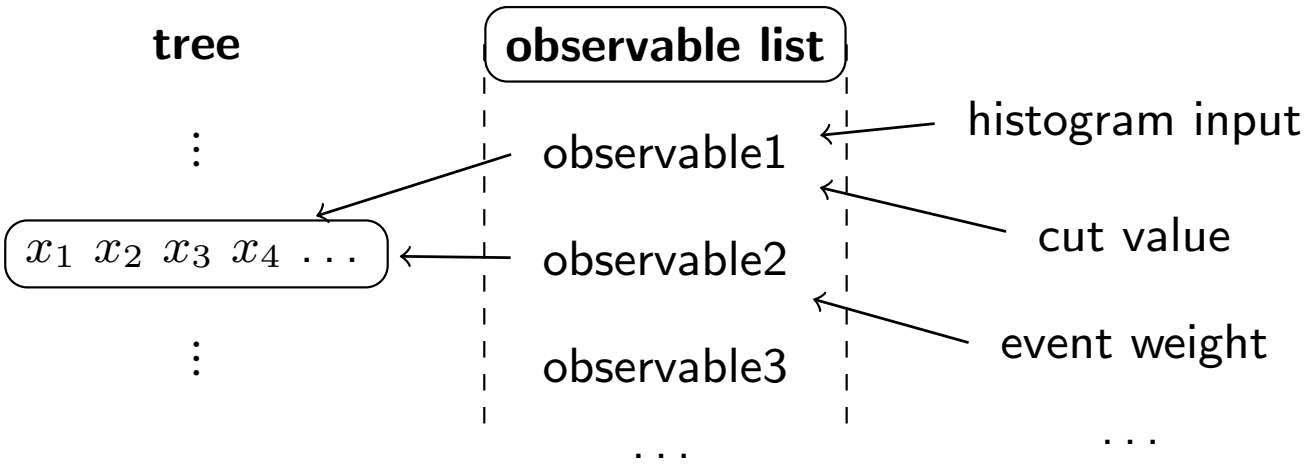
nTuples and xAODs

State of the Art

Above and Beyond

Dos & Don'ts

Observables



- observables are identified by their **expression**
- expressions can be arithmetic expressions, but also names
- observables can be obtained and created by `TQObservable::getObservable`
- can also manually add observables to the list with `TQObservable::addObservable`

Introduction
Booking Histograms
nTuples and xAODs
State of the Art
Above and Beyond
Dos & Don'ts

Histogramming

Introduction
Booking Histograms
nTuples and xAODs
State of the Art
Above and Beyond
Dos & Don'ts

Run 1

```
TH1F('Mll', '', 60, 0., 120.) << ( Mll/1000 : 'm_{ll} [GeV]' );
```

Run 2

```
TH1F('Mll', '', 60, 0., 120.) << ( HMMCands[0].constituent(0).m()/1000 : 'm_{ll} [GeV]' );
```

- all that changed is this:
  `Mll` → `HMMCands[0].constituent(0).m()`
- what does this mean?

Flatness and Complexity

Introduction
Booking Histograms
nTuples and xAODs
State of the Art
Above and Beyond
Dos & Don'ts

Flat nTuple (`TTree`)

- branch1 (`float`)
- branch2 (`float`)
- branch3 (`int`)
- ...
- `Mll` (`float`)
- ...

xAOD (`TTree`)

- branch1 (`TruthParticleContainer`)
- branch2 (`IParticleContainer`)
- branch3 (`MyCustomObjContainer`)
- ...
- `HMMCands` (`IParticleContainer`)
- ...

Complexity and $m_{\ell\ell}$

Introduction
Booking Histograms
nTuples and xAODs
State of the Art
Above and Beyond
Dos & Don'ts

```
HMMCands[0].constituent(0).m()
```

- `HMMCands[0]` is a Higgs candidate, an object of type `IParticle`, subtype `CompositeParticle`
- composite particles have constituents, which are also of type `IParticle`
- they can be accessed with the member function `CompositeParticle::constituent(int n)`
- the argument n is the index of the constituent
- `HMMCands[0].constituent(0)` is also a `CompositeParticle`: the dilepton object
- `IParticles` (and thus also `CompositeParticle`) have non-argument member function `m()` to access the mass
- thus, `HMMCands[0].constituent(0).m()` is $m_{\ell\ell}$

High hopes

Introduction
Booking Histograms
nTuples and xAODs
State of the Art
Above and Beyond
Dos & Don'ts

- xAOD EDM objects can have *arbitrary* member functions
- ask the leading electron in $H \to WW \to 2e2\nu$ candidates if it passes the `Loose` selection:

```
HEECands[0].constituent(0).electron(0).passSelection("Loose")
```

It's wonderful, isn't it?

Histogramming

- we start by calling `TQHistoMakerAnalysisJob::importJobsFromTextFiles` on our file

`TQHistoMakerAnalysisJob::importJobsFromTextFil`

- read in your text file line by line
- create a `TQHistoMakerAnalysisJob` for you
- book each single histogram you requested calling `TQHistoMakerAnalysisJob::bookHistogram`

- we have created a histogram job
- we will now see how histogram booking works

Introduction
State of the Art
Understand the Process
Understand the Issues
Observable Types
Status Summary
Above and Beyond
Dos & Don'ts

Booking Histograms

- a histogram booking has been requested

TQHistoMakerAnalysisJob::bookHistogram

- read the definition, the expression and the title

```
TH1F('Mll', '', 60, 0., 120.)  << ( Mll :  'm_ll [GeV]' );
```

- request a TQObservable for this expression calling TQObservable::getObservable

- histogram jobs don't know about trees or data
- all data retrieval is done by observables
- we will now see how they work

Introduction
State of the Art
Understand the Process
Understand the Issues
Observable Types
Status Summary
Above and Beyond
Dos & Don'ts

Retrieving Observables

- an observable has been requested

`TQObservable::getObservable("Mll/1000.")`

- recognize that this is a *formula*-like `expression`
- convert it into a *name* (no slashes: `"Mll:1000.`)
- check the global list of observables if there already is an observable with matching *name*
- if a match is found, return it
- otherwise, create one by creating a `TQTreeFormulaObservable` (special type of `TQObservable`) with this *name* and *expression*

Introduction
State of the Art
Understand the Process
Understand the Issues
Observable Types
Status Summary
Above and Beyond
Dos & Don'ts

TQTreeFormulaObservable

TQTreeFormulaObservable members:

- fTree (TTree*): , pointer to tree
- fExpression (TString): constructor argument
- fFormula (TTreeFormula): ROOT class, needs fTree and fTree. does all the hard work.

TQTreeFormulaObservable methods:

- initializeSelf: for each sample, create fFormula
- getValue: on each event, return fFormula->Eval
- finalizeSelf: after finishing, delete the fFormula

Introduction
State of the Art
Understand the Process
Understand the Issues
Observable Types
Status Summary
Above and Beyond
Dos & Don'ts

Problems

- constructor of `TTreeFormula` is called with arguments
  - ✓ `Mll`
  - ✓ `HEECands[0].constituent(0).m()`
  - ✗ `HEECands[0].constituent(0).electron(0).passSelection("Loose")`
- `TTreeFormula` was never designed to do any of this:
  - ✓ handle branches that are objects (not simple data types)
  - ✓ call methods of these objects
  - ✓ pass arguments these methods
  - ✗ create other objects on-the-fly to pass them to these methods as arguments
- rewrite of `TTreeFormula` along with `ROOT6` upcoming
- will be too late for first Run 2 data (Fall '15 earliest)

Introduction
State of the Art
Understand the Process
Understand the Issues
Observable Types
Status Summary
Above and Beyond
Dos & Don'ts

Examples

✓ `HEECands[0].constituent(0).m()`

- branch is of object type,object methods are called
- only integers are passed to these methods as arguments

✗ `HEECands[0].constituent(0).electron(0).passSelection("Loose")`

- a string (object-type) is passed as an argument

✗ `HEECands[0].deltaPhi({1,2})`

- a list (object-type, C++11 syntax) is passed

✓ `HEECands[0].deltaPhi(1,2)`

- only two integers are passed as arguments

? `HEECands[0].deltaPhi(HEECands[0].constituent(0),HEECands[1].constituent(1))`

- pointers to objects are passed as arguments

Introduction
State of the Art
Understand the Process
Understand the Issues
Observable Types
Status Summary
Above and Beyond
Dos & Don'ts

Status of the Problem

- some types of data cannot be accessed with `TTreeFormula`
- in some cases, hacky workarounds might exist
- in general, a clean method of accessing tree data is required

Reminder: TQObservables

- `TTreeFormula` is the access method used by `TQTreeFormulaObservable`
- this is only **one type** of `TQObservable`, others exist
- we will now get to know them

Introduction
State of the Art
Understand the Process
Understand the Issues
Observable Types
Status Summary
Above and Beyond
Dos & Don'ts

TQTreeFormulaObservable

Introduction
State of the Art
Understand the Process
Understand the Issues
Observable Types
Status Summary
Above and Beyond
Dos & Don'ts

characteristic	evaluates a `TTreeFormula` on every event
purpose	bread-and-butter physics
complexity	low
usage	default observable type, used unless specified otherwise

TQConstObservable

Introduction
State of the Art
Understand the Process
Understand the Issues
Observable Types
Status Summary
Above and Beyond
Dos & Don'ts

characteristic returns the same value for every event in a sample

purpose performance gain for trivial conditions like $1 == 1$

complexity trivial

usage is automatically used for expressions that don't contain any letters, e. g. `1+1 == 2`

TQMVAObservable

Introduction
State of the Art
Understand the Process
Understand the Issues
Observable Types
Status Summary
Above and Beyond
Dos & Don'ts

characteristic retrieves data from an instance of `TMVA`

purpose on-the-fly evaluation of BDTs etc.

complexity high

usage automatically used for expressions that contain the string "weights.xml". the expression used as a filename to read the TMVA configuration from.

TQMultiObservable

Introduction
State of the Art
Understand the Process
Understand the Issues
Observable Types
Status Summary
Above and Beyond
Dos & Don'ts

characteristic retrieves data from other observables (not from the tree)

purpose allows for more complex calculatons

complexity medium

usage is automatically used for expressions that contain names enclosed in brackets, e. g.
`[myMVAweights.xml] > 0.5`
but not
`HMMCands[0]`

Summary

Introduction
State of the Art
Understand the Process
Understand the Issues
Observable Types
Status Summary
Above and Beyond
Dos & Don'ts

- we need observabless to retrieve data from the samples for histograms, cuts, and everything else
- neither of the pre-existing observable types supports the full list of xAOD features
- only the `TQTreeFormulaObservable` *actually* retrieves data from the tree

Your new Observable type

What is an observable?

- a class inheriting from `TQObservable`
- that implements a method `getValue`
- that returns a *value* for every event

What can observables do?

- access any data from the tree
- execute any code you choose to write

Introduction
State of the Art
Above and Beyond
Overview
Coding Guide
TEvent
TQEventObservable
TQTreeObservable
wizard.py
Dos & Don'ts

Creating your observable

Write your class

- create `MyObservable.cxx` and `MyObservable.h`
- have your class inherit from `TQObservable`
- implement `getValue`

- create an instance of your class
  `myObs = MyObservable()`
- append your observable to the list and assign a name
  `TQObservable::addObservable(myObs,"XYZ")`
- use it in your histogram or cut definition files

```
TH1F('xyz, '', 60, 0., 120.)  << ( XYZ : 'My Observable' );
```

Introduction
State of the Art
Above and Beyond
Overview
Coding Guide
TEvent
TQEventObservable
TQTreeObservable
wizard.py
Dos & Don'ts

Access Mechanisms

For your code, you can use various access mechanisms:

- `TTreeFormula` – useless
  - Pro easy to use
  - Con not feature-complete, hence pointless
- `TTree::SetBranchAddress` – discouraged
  - Pro easy to use, well known, highly performant
  - Con address ownership is not tracked, different observables trying to access the same branch will interfere
- `xAOD::TEvent` – recommended
  - Pro highly performant, feature-complete
  - Con requires custom coding, (currently) no way to autocreate from text files

Introduction
State of the Art
Above and Beyond
Overview
Coding Guide
TEvent
TQEventObservable
TQTreeObservable
wizard.py
Dos & Don'ts

Inherit from one of three base types

`TQEventObservable` uses `xAOD::TEvent`

- officially recommended
- works on xAOD only
- example: `TQTreeFormulaObservable`

`TQTreeObservable` uses `TTree`

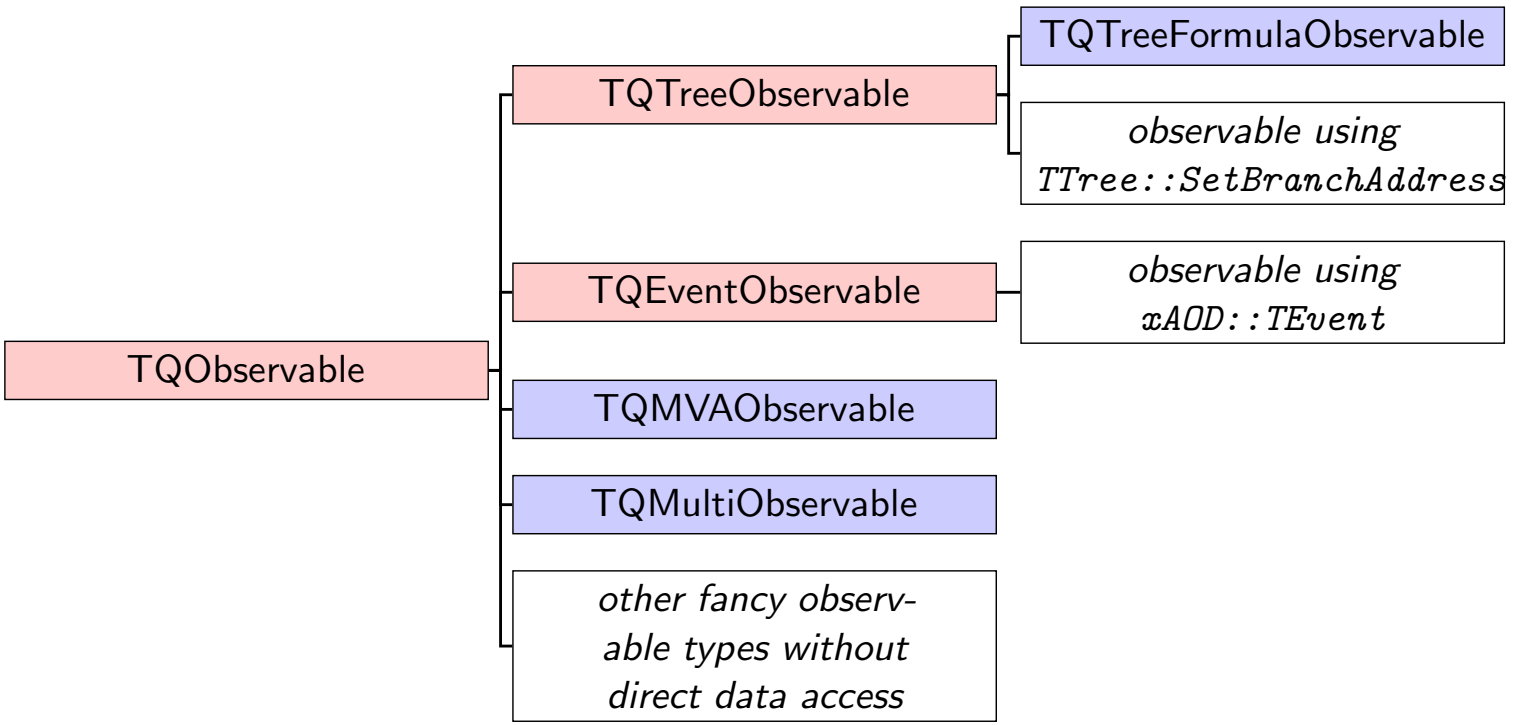
- use for any standard-ROOT mechanism
- works well on flat nTuples
- does not support all xAOD features

`TQObservable` no predefined mechanism. useful if you

- use other observables as input. ex.: `TQMVAObservable`, `TQMultiObservable`
- read data from separate files
- use randomly generated numbers
- use constant values. ex.: `TQConstObservable`
- ...

Introduction
State of the Art
Above and Beyond
Overview
Coding Guide
TEvent
TQEventObservable
TQTreeObservable
wizard.py
Dos & Don'ts

Inheritance structure



Introduction
State of the Art
Above and Beyond
Overview
Coding Guide
TEvent
TQEventObservable
TQTreeObservable
wizard.py
Dos & Don'ts

xAODs: Truth uncovered

What you think

- branch1 (`TruthParticleContainer`)
- branch2 (`IParticleContainer`)
- branch3 (`MyCustomObjContainer`)
- ...

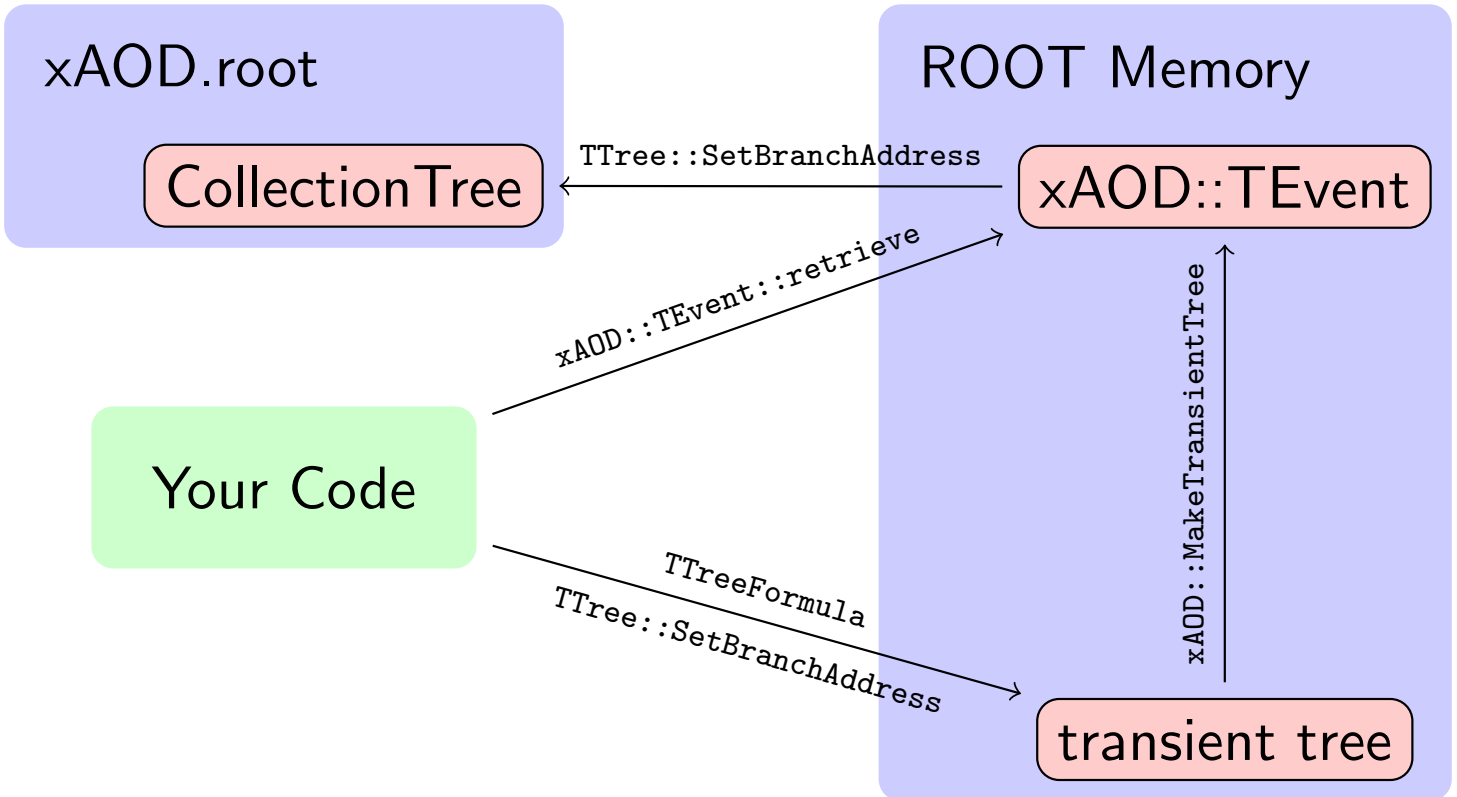
How it is

- branch1Aux.e (`float`)
- branch1Aux.px (`float`)
- branch1Aux.py (`float`)
- branch1Aux.pz (`float`)
- branch2Aux.e (`float`)
- ...

- how do we get from right to left?
- how do we actually get *class-type* objects in branches?

Introduction
State of the Art
Above and Beyond
Overview
Coding Guide
TEvent
TQEventObservable
TQTreeObservable
wizard.py
Dos & Don'ts

xAODs: How it works



- inherit from `TQTreeObservable` to access transient tree
- inherit from `TQEventObservable` to access `xAOD::TEvent`

Introduction
State of the Art
Above and Beyond
Overview
Coding Guide
TEvent
TQEventObservable
TQTreeObservable
wizard.py
Dos & Don'ts

Writing a TQEventObservable

- inherit from `TQEventObservable`
- implement `MyObservable::getValue`
- use `xAOD::TEvent::retrieve` to access data
- return your result

```
class MyObservable : public TQEventObservable{
protected:
  mutable xAOD::CompositeParticleContainer const * mCand = 0;
public:
  double MyObservable::getValue() const override {
    if(!this->fEvent->retrieve(this->mCand, "EECands").isSuccess()) return 0;
    const xAOD::CompositeParticle* p = this->mCand->at(0);
    return p->electron(0)->passSelection("Loose");
  }
};
```

Introduction
State of the Art
Above and Beyond
Overview
Coding Guide
TEvent
TQEventObservable
TQTreeObservable
wizard.py
Dos & Don'ts

Writing a TQTreeObservable

- inherit from TQTreeObservable
- implement getValue, initializeSelf, finalizeSelf and getBranchNames
- use TTreeFormula to retrieve data

```
class MyObservable : public TQTreeObservable{
protected:
  TTreeFormula* fFormula = 0;
public:
  bool MyObservable::initializeSelf() override {
    this->fFormula = new TTreeFormula("EECands.constituent(0).m()",this->fTree);
    return true;
  }
  bool MyObservable::finalizeSelf() override {
    delete this->fFormula;
    return true;
  }
  double MyObservable::getValue() const override {
    return this->fFormula->Eval(0.);
  }
  TObjArray * getBranchNames(TQSample *s) const override {
    TObjArray* retval = new TObjArray();
    retval->Add(new TObjString("EECands"));
    return retval;
  }
};
```


Introduction
State of the Art
Above and Beyond
Overview
Coding Guide
TEvent
TQEventObservable
TQTreeObservable
wizard.py
Dos & Don'ts

Writing a TQTreeObservable

- inherit from TQTreeObservable
- implement getValue, initializeSelf, finalizeSelf and getBranchNames
- use TTree::SetBranchAddress to retrieve data

```
class MyObservable : public TQTreeObservable{
protected:
  mutable xAOD::CompositeParticleContainer const * mCand = 0;
public:
  bool MyObservable::initializeSelf() override {
    this->fTree->SetBranchAddress("EECands",&(this->mCand)); // please don't do this
    return true;
  }
  bool MyObservable::finalizeSelf() override {
    return true;
  }
  double MyObservable::getValue() const override {
    return this->mCand->constituent(0)->m();
  }
  TObjArray * getBranchNames(TQSample *s) const override {
    TObjArray* retval = new TObjArray();
    retval->Add(new TObjString("EECands"));
    return retval;
  }
};
```

Introduction
State of the Art
Above and Beyond
Overview
Coding Guide
TEvent
TQEventObservable
TQTreeObservable
wizard.py
Dos & Don'ts

Copy & paste, search & replace

Introduction
State of the Art
Above and Beyond
Overview
Coding Guide
TEvent
TQEventObservable
TQTreeObservable
wizard.py
Dos & Don'ts

- you can copy & paste many features from existing observables or this talk
- but you don't have to!

`QFramework/share/templates/TQObservable/wizard.py`

- semi-intelligent wizard will guide you through the process of writing your observable
- will provide you with commented templates for all three observables types
- will automatically create `LinkDef` entries & move your files to the correct location

TQEventObservable

Introduction
State of the Art
Above and Beyond
Dos & Don'ts
Coding
Usage

- ✓ Check the return value of `xAOD::TEvent::retrieve`
  `if(!this->fEvent->retrieve(this->mCand, "EECands").isSuccess()) return 0;`
- ✗ Forgetting to do this will produce warnings about unchecked return codes!
- ✓ optimize implementation of `getValue` – it will be called multiple times per event!
- ✓ use the keywords `const` and `override` – they help you avoid bugs!

TQTreeObservable

Introduction
State of the Art
Above and Beyond
Dos & Don'ts
Coding
Usage

- ✓ feel free to use `TTreeFormula` whenever applicable
- ✓ create all transient data members in `initializeSelf`
- ✗ don't forget to undo what you did in `finalizeSelf`. forgetting to delete members here will produce memory leaks!
- ✗ **please don't** use `TTree::SetBranchAddress`, unless you are **very sure** that **nobody else will ever use the same branch!**

Caution: Observables are not branches!

Introduction
State of the Art
Above and Beyond
Dos & Don'ts
Coding
Usage

✗ `TH1F('xyz, '', 60, 0., 120.) << ( XYZ/1000. : 'My Observable' );`

- your observable name is "XYZ", not "XYZ/1000."
- here, the framework will fail to find your observable and try to build a `TQTreeFormulaObservable` instead
- this in turn will fail to find a branch named "XYZ" and produce error messages

✓ `TH1F('xyz, '', 60, 0., 120.) << ( [XYZ]/1000. : 'My Observable' );`

- the brackets will trigger the creation of a `TQMultiObservable`
- that in turn will ask for your observable, find it, evaluate it, and evaluate the expression using its output

Caution: Observables are not branches!

✗ `TH1F('xyz, '', 60, 0., 120.) << ( [XYZ]/1000.+lepPt1 : 'My Observable' );`

- the brackets will trigger the creation of a `TQMultiObservable`
- but this one only knows how to handle observables, not branches
- you will get error messages complaining about invalid numerical expressions

✓ `TH1F('xyz, '', 60, 0., 120.) << ( [XYZ]/1000.+[lepPt1] : 'My Observable' );`

- the brackets will trigger the creation of a `TQMultiObservable`
- this will work on two observables, named "XYZ" and "lepPt1".
- the first of these two will be your predefined observable
- the second one will be created as a `TQTreeFormulaObservable`

Introduction
State of the Art
Above and Beyond
Dos & Don'ts
Coding
Usage

Summary

Introduction
State of the Art
Above and Beyond
Dos & Don'ts
Coding
Usage

- the CAF is a powerful framework. it can do almost anything – if you know how!
- xAODs provide a convenient data model – but some features are still missing
- we can compensate this by writing customized observables accessing the xAOD content
- don't be afraid to write your own observable class – it's easy and useful
- don't hesitate to contact me for remaining questions or ask for help