RAPIDO

Generated by Doxygen 1.9.2

Chapter 1

RAPIDO

Repeatable Analysis Programming for Interpretability, Durability, and Organization

RAPIDO is a C++ framework designed to make writing HEP analyses more ergonomic and readable. It wraps some basic functionality of `ROOT`. The idea is that an analysis, in general, consists of a few objects: a TTree (to hold some skimmed N-Tuple and/or a set of histograms), a cutflow (a collection of boolean logic for filtering events), and a looper (some way to run over multiple files). RAPIDO is designed to handle all three of these tasks such that every analysis that uses it is structured in the same way. In addition, the *way* in which it is structured lends itself to the common workflow of a HEPEx-er.

1.1 RAPIDO Tools

1. Arbol: TTree wrapper that reduces the hassle of setting up and using TTrees
2. Cutflow: Binary search tree with lambda nodes and other bells and whistles
    - Histflow: An extension of the Cutflow object that handles histogramming at any given step of the cutflow
3. Looper: Basic looper for a TChain of TFiles that uses any selector

1.2 Set Up Instructions

1. Clone this repository
2. `cd` into the cloned repository and run `make -j5`
3. Write your script (e.g. `main.cc`) and `#include` whatever you need
4. Compile and run using your favorite `Makefile`:

```
$ make
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PWD/rapido/src
$ export ROOT_INCLUDE_PATH=$ROOT_INCLUDE_PATH:$PWD/rapido/src
$ ./main
```

1.3 Examples

1. Minimal Cutflow example

```
#include "cutflow.h"
#include <stdlib.h>
using namepsace std;
int main()
{
    Cutflow dummy_cutflow = Cutflow();
    Cut* dummy_root = new Cut("root", []() { return bool(rand() % 2); });
    dummy_cutflow.setRoot(dummy_root);
    Cut* node0 = new Cut("node0", []() { return bool(rand() % 2); });
    dummy_cutflow.insert("root", node0, Left);
    Cut* node1 = new Cut("node1", []() { return bool(rand() % 2); });
    dummy_cutflow.insert("root", node1, Right);
    Cut* node2 = new Cut("node2", []() { return bool(rand() % 2); });
    dummy_cutflow.insert("node1", node2, Right);
    Cut* node3 = new Cut("node3", []() { return bool(rand() % 2); });
    dummy_cutflow.insert("node1", node3, Left);
    Cut* node4 = new Cut("node4", []() { return bool(rand() % 2); });
    dummy_cutflow.insert("node2", node4, Right);
    for (int i = 0; i < 5; i++)
    {
        Cut* terminal_node = dummy_cutflow.run();
        cout « "terminated at " « terminal_node->name « endl;
    }
    dummy_cutflow.print();
    return 0;
}
```

2. A simple Arbol+Looper example (using `ROOT::MakeSelector` to read an arbitrary ROOT file)

```
$ root # only known to work for ROOT v6.22 and greater
root[0] TFile* f = new TFile("/path/to/myfile.root")
root[1] TreeName->MakeSelector("MySelector")
(int) 0
root [2] .q
$ mv MySelector.C MySelector.cc
$ rootcint myselectordict.cc -c MySelector.h
$ mv myselectordict* rapdio/
$ mv MySelector* rapido/
$ cd rapido/
$ make clean
$ make -j5
// Selector
#include "MySelector.h"
// RAPIDO
#include "arbol.h"
#include "looper.h"
int main()
{
    // Initialize Arbol
    TFile* output_tfile = new Tfile("output.root", "RECREATE");
    Arbol arbol = Arbol(output_tfile);
    // Initialize branches
    arbol.newBranch<int>("event");
    arbol.newBranch<float>("met");
    arbol.newBranch<float>("ht");
    arbol.newBranch<int>("n_jets");
    arbol.newVecBranch<float>("good_jet_pt"); // newVecBranch<float> <--> newBranch<std::vector<float»
    // Get file
    TChain* tchain = new TChain("TreeName");
    tchain->Add("/path/to/myfile.root");
    // Initialize Looper
    MySelector selector;
    Looper looper = Looper<Nano>(&selector, tchain, "TreeName");
    // Run
    looper.run(
        [&](TTree* ttree) { selector.Init(ttree) },
        [&](int entry)
        {
            selector.GetEntry(entry);
            selector.Process(entry);
            // --> Event-level Logic <--
            // Reset tree
            arbol.resetBranches(); // variables like arbol and selector are captured by reference
            // Loop over jets
            float ht = 0.;
            for (unsigned int i = 0; i < *selector.nJet; i++)
            {
                if (selector.Jet_pt[i] > 30)
                {
                    arbol.appendToVecLeaf<float>("good_jet_pt", selector.Jet_pt[i]);
                    ht += selector.Jet_pt[i];
                }
            }
```

```
                arbol.setLeaf<int>("event", *selector.event);
                arbol.setLeaf<float>("ht", ht);
                arbol.setLeaf<float>("met", *selector.MET_pt);
                arbol.setLeaf<int>("n_jets", arbol.getVecLeaf<float>("goot_jet_pt").size());
                arbol.fillTTree();
                return;
            }
        );
        // Write results to a ROOT file
        arbol.writeTFile();
        return 0;
    }
```

3. Arbol+Cutflow+Looper+HEPCLI example (now using `NanoCORE` to read NanoAOD)

```
    // ROOT
    #include "TH1F.h"
    // NanoCORE
    #include "Nano.h"
    #include "tqdm.h" // progress bar
    #include "SSSelections.h"
    #include "ElectronSelections.h"
    #include "MuonSelections.h"
    // RAPIDO
    #include "arbol.h"
    #include "cutflow.h"
    #include "looper.h"
    using namespace std;
    using namespace tas;
    int main(int argc, char** argv)
    {
        // CLI
        HEPCLI cli = HEPCLI(argc, argv);
        // Initialize Looper
        Looper looper = Looper<Nano>(&nt, cli.input_tchain);
        // Initialize Arbol
        Arbol arbol = Arbol(cli.output_tfile);
        // Event branches
        arbol.newBranch<int>("event", -999);
        arbol.newBranch<float>("met", -999);
        // Leptons
        arbol.newBranch<int>("leading_lep_id", -999);
        arbol.newBranch<float>("leading_lep_pt", -999);
        arbol.newBranch<float>("leading_lep_eta", -999);
        arbol.newBranch<float>("leading_lep_phi", -999);
        arbol.newBranch<int>("trailing_lep_id", -999);
        arbol.newBranch<float>("trailing_lep_pt", -999);
        arbol.newBranch<float>("trailing_lep_eta", -999);
        arbol.newBranch<float>("trailing_lep_phi", -999);
        // Initialize Cutflow
        Cutflow cutflow = Cutflow();
        // Initialize some hists
        TH1F* ld_lep_pt_hist = new TH1F("ld_lep_pt_hist", "ld_lep_pt_hist", 20, 0, 200);
        TH1F* tr_lep_pt_hist = new TH1F("tr_lep_pt_hist", "tr_lep_pt_hist", 20, 0, 200);
        cutflow.globals.newVar<TH1F>("ld_lep_pt_hist", *ld_lep_pt_hist);
        cutflow.globals.newVar<TH1F>("tr_lep_pt_hist", *tr_lep_pt_hist);
        Cut* root = new Cut(
            "Bookkeeping",
            [&]()
            {
                arbol.setLeaf("event", nt.event());
                arbol.setLeaf("met", nt.MET_pt());
                return true;
            },
            [&]()
            {
                // Dummy weight
                return 0.001;
            }
        );
        cutflow.setRoot(root);
        Cut* dilep_presel = new Cut(
            "DileptonPreselection",
            [&]()
            {
                int n_tight_leps = 0;
                int n_loose_not_tight_leps = 0;
                Leptons leptons = getLeptons();
                Lepton leading_lep;
                Lepton trailing_lep;
                for (auto& lep : leptons)
                {
                    if (lep.pt() < 20) { continue; }
                    if (lep.idlevel() == SS::IDtight)
                    {
                        if (lep.pt() > leading_lep.pt())
                        {
                            trailing_lep = leading_lep;
                            leading_lep = lep;
                        }
                        else if (lep.pt() > trailing_lep.pt()) { trailing_lep = lep; }
                        n_tight_leps++;
                    }
                    if (lep.idlevel() == SS::IDfakable) { n_loose_not_tight_leps++; }
                }
                if (n_tight_leps == 2 && n_loose_not_tight_leps == 0)
                {
                    arbol.setLeaf<int>("leading_lep_id", leading_lep.id());
                    arbol.setLeaf<float>("leading_lep_pt", leading_lep.pt());
                    arbol.setLeaf<float>("leading_lep_eta", leading_lep.eta());
                    arbol.setLeaf<float>("leading_lep_phi", leading_lep.phi());
                    arbol.setLeaf<int>("trailing_lep_id", trailing_lep.id());
                    arbol.setLeaf<float>("trailing_lep_pt", trailing_lep.pt());
                    arbol.setLeaf<float>("trailing_lep_eta", trailing_lep.eta());
                    arbol.setLeaf<float>("trailing_lep_phi", trailing_lep.phi());
                    return true;
                }
                else { return false; }
            }
        );
        cutflow.insert("Bookkeeping", dilep_presel, Right);
        Cut* monolep_or_fakes = new Cut("SingleLepOrFakes", [&]() { return true; });
        cutflow.insert("DileptonPreselection", monolep_or_fakes, Left);
        Cut* dilep_sign = new Cut(
            "CheckDilepSign",
            [&]()
            {
                int leading_lep_id = arbol.getLeaf<int>("leading_lep_id");
                int trailing_lep_id = arbol.getLeaf<int>("trailing_lep_id");
                return leading_lep_id*trailing_lep_id > 0;
            }
        );
        cutflow.insert("DileptonPreselection", dilep_sign, Right);
        Cut* SS_presel = new Cut("SSPreselection", [&]() { return true; });
        cutflow.insert("CheckDilepSign", SS_presel, Right);
        Cut* OS_presel = new Cut(
            "OSPreselection",
            [&]()
            {
                TH1F& ld_lep_pt_hist = cutflow.globals.getRef<TH1F>("ld_lep_pt_hist");
                TH1F& tr_lep_pt_hist = cutflow.globals.getRef<TH1F>("tr_lep_pt_hist");
                ld_lep_pt_hist.Fill(arbol.getLeaf<float>("leading_lep_pt"));
                tr_lep_pt_hist.Fill(arbol.getLeaf<float>("trailing_lep_pt"));
                return true;
            },
            [&]()
            {
                // Dummy weight
                return 0.25;
            }
        );
        cutflow.insert("CheckDilepSign", OS_presel, Left);
        // Run looper
        tqdm bar; // progress bar
        looper.run(
            [&](TTree* ttree)
            {
                nt.Init(ttree);
            },
            [&](int entry)
            {
                bar.progress(looper.n_events_processed, looper.n_events_to_process);
                nt.GetEntry(entry);
                // Reset tree
                arbol.resetBranches();
                // Run cutflow
                bool passed = cutflow.runUntil("OSPreselection");
                if (passed) { arbol.fillTTree(); }
                return;
            }
        );
        // Wrap up
        bar.finish();
        cutflow.print();
        cutflow.writeCSV();
        arbol.writeTFile();
        return 0;
    }
```

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 Arbol Class Reference

```
#include <arbol.h>
```

Public Member Functions

- Arbol ()
- Arbol (TFile ∗new_tfile)
- virtual ∼Arbol ()
- template<typename Type >
  void newBranch (TString new_branch_name)
- template<typename Type >
  void newBranch (TString new_branch_name, Type new_reset_value)
- template<typename Type >
  void setBranchResetValue (TString branch_name, Type new_reset_value)
- template<typename Type >
  Type getLeaf (TString branch_name)
- template<typename Type >
  void setLeaf (TString branch_name, Type new_value)
- template<typename Type >
  void newVecBranch (TString new_branch_name)
- template<typename Type >
  void newVecBranch (TString new_branch_name, std::vector< Type > new_reset_vector)
- template<typename Type >
  void setVecBranchResetValue (TString branch_name, std::vector< Type > new_reset_vector)
- template<typename Type >
  std::vector< Type > getVecLeaf (TString branch_name)
- template<typename Type >
  void setVecLeaf (TString branch_name, std::vector< Type > new_vector)
- template<typename Type >
  void appendToVecLeaf (TString branch_name, Type new_value)
- template<typename Type >
  void prependToVecLeaf (TString branch_name, Type new_value)
- template<typename Type >
  void insertIntoVecLeaf (TString branch_name, Type new_value, int index)
- template<typename Type >
  void sortVecLeaf (TString branch_name, std::function< bool(Type, Type)> &lambda)
- void resetBranches ()
- void fillTTree ()
- void writeTFile ()

Public Attributes

- TTree ∗ ttree
- TFile ∗ tfile

Protected Member Functions

- template<typename Type >
  Branch< Type > ∗ getBranch (TString branch_name)

Protected Attributes

- std::map< TString, Utilities::Dynamic ∗ > branches
- std::map< TString, std::function< void()> > branch_resetters

5.1.1 Detailed Description

Wraps TTree object with functionality for making branches dynamically

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Arbol() [1/2]

```
Arbol::Arbol ( )
```

Arbol object constructor

Returns

none

5.1.2.2 Arbol() [2/2]

```
Arbol::Arbol (
             TFile * new_tfile )
```

Arbol object overload constructor

Parameters

new_tfile	pointer to an output TFile

Returns

none

5.1.2.3 ∼Arbol()

```
virtual Arbol::∼Arbol ( )  [virtual]
```

Arbol object destructor

Returns

none

5.1.3 Member Function Documentation

5.1.3.1 appendToVecLeaf()

```
template<typename Type >
void Arbol::appendToVecLeaf (
             TString branch_name,
             Type new_value )
```

Append given value to leaf (vector)

Template Parameters

Type	type of branch value

Parameters

branch_name	branch name
new_value	new value to append

Returns

none

5.1.3.2 fillTTree()

```
void Arbol::fillTTree ( )
```

Fill TTree with all current leaves

Returns

none

5.1.3.3 getBranch()

```
template<typename Type >
Branch< Type > * Arbol::getBranch (
             TString branch_name )  [protected]
```

(PROTECTED) Get pointer to branch object if it exists

Template Parameters

Type	type of branch value

Parameters

branch_name	branch name

Returns

pointer to branch object

5.1.3.4 getLeaf()

```
template<typename Type >
Type Arbol::getLeaf (
             TString branch_name )
```

Get current leaf value

Template Parameters

Type	type of branch value

Parameters

branch_name	branch name

Returns

leaf value

5.1.3.5 getVecLeaf()

```
template<typename Type >
std::vector< Type > Arbol::getVecLeaf (
             TString branch_name )
```

Calls Arbol::getLeaf, but supplies std::vector<Type> for tparam

See also

Arbol::getLeaf

Template Parameters

Type	type of branch value

Parameters

branch_name	branch name

Returns

leaf vector

5.1.3.6 insertIntoVecLeaf()

```
template<typename Type >
void Arbol::insertIntoVecLeaf (
             TString branch_name,
             Type new_value,
             int index )
```

Insert value into leaf (vector) at a particular index

Template Parameters

Type	type of branch value

Parameters

branch_name	branch name
new_value	new value to insert
index	target index

Returns

none

5.1.3.7 newBranch() [1/2]

```
template<typename Type >
void Arbol::newBranch (
             TString new_branch_name )
```

Add a new branch to TTree

Template Parameters

Type	type of branch value

Parameters

new_branch_name	new branch name

Returns

none

5.1.3.8 newBranch() [2/2]

```
template<typename Type >
void Arbol::newBranch (
             TString new_branch_name,
             Type new_reset_value )
```

Add a new branch to TTree and set reset value

Template Parameters

Type	type of branch value

Parameters

new_branch_name	new branch name
new_reset_value	new branch reset value

Returns

none

5.1.3.9 newVecBranch() [1/2]

```
template<typename Type >
void Arbol::newVecBranch (
             TString new_branch_name )
```

Calls Arbol::newBranch, but supplies std::vector<Type> for tparam

See also

Arbol::newBranch

Template Parameters

Type	type of vector branch value

Parameters

new_branch_name	branch name

Returns

none

5.1.3.10 newVecBranch() [2/2]

```
template<typename Type >
void Arbol::newVecBranch (
             TString new_branch_name,
             std::vector< Type > new_reset_vector )
```

Calls Arbol::newBranch, but supplies std::vector<Type> for tparam

See also

Arbol::newBranch

Template Parameters

Type	type of vector branch value

Parameters

new_branch_name	new branch name
new_reset_vector	new branch reset value (vector)

Returns

none

5.1.3.11 prependToVecLeaf()

```
template<typename Type >
void Arbol::prependToVecLeaf (
             TString branch_name,
             Type new_value )
```

Prepend given value to leaf (vector)

Template Parameters

Type	type of branch value

Parameters

branch_name	branch name
new_value	new value to prepend

Returns

none

5.1.3.12 resetBranches()

```
void Arbol::resetBranches ( )
```

Set value of each branch to its respective reset value Uses a map of "resetters" for the same reason as Utilities::Variables.

Returns

none

5.1.3.13 setBranchResetValue()

```
template<typename Type >
void Arbol::setBranchResetValue (
             TString branch_name,
             Type new_reset_value )
```

Set reset value for the branch

Template Parameters

Type	type of branch value

Parameters

branch_name	branch name
new_reset_value	new reset value

Returns

none

5.1.3.14 setLeaf()

```
template<typename Type >
void Arbol::setLeaf (
             TString branch_name,
             Type new_value )
```

Set current leaf value

Template Parameters

Type	type of branch value

Parameters

branch_name	branch name
new_value	new value

Returns

none

5.1.3.15 setVecBranchResetValue()

```
template<typename Type >
void Arbol::setVecBranchResetValue (
             TString branch_name,
             std::vector< Type > new_reset_vector )
```

Calls Arbol::setBranchResetValue, but supplies std::vector<Type> for tparam

See also

Arbol::setBranchResetValue

Template Parameters

Type	type of vector branch value

Parameters

branch_name	branch name
new_reset_vector	new branch reset value (vector)

Returns

none

5.1.3.16 setVecLeaf()

```
template<typename Type >
void Arbol::setVecLeaf (
             TString branch_name,
             std::vector< Type > new_vector )
```

Calls Arbol::setLeaf, but supplies std::vector<Type> for tparam

See also

Arbol::getLeaf

Template Parameters

Type	type of branch value

Parameters

branch_name	branch name
new_vector	new branch value (vector)

Returns

none

5.1.3.17 sortVecLeaf()

```
template<typename Type >
void Arbol::sortVecLeaf (
            TString branch_name,
            std::function< bool(Type, Type)> & lambda )
```

Sort leaf (vector) using a given lambda function

Template Parameters

Type	type of branch value

Parameters

branch_name	branch name
lambda	lambda function to use for sorting

Returns

none

5.1.3.18 writeTFile()

```
void Arbol::writeTFile ( )
```

Write TTree to TFile

Returns

none

5.1.4 Member Data Documentation

5.1.4.1 branch_resetters

```
std::map<TString, std::function<void()> > Arbol::branch_resetters  [protected]
```

Map of reset function for each dynamically typed TBranch

5.1.4.2 branches

```
std::map<TString, Utilities::Dynamic*> Arbol::branches  [protected]
```

Map of dynamically typed TBranches

5.1.4.3 tfile

```
TFile* Arbol::tfile
```

Pointer to ROOT TFile object

5.1.4.4 ttree

```
TTree* Arbol::ttree
```

Pointer to ROOT TTree object

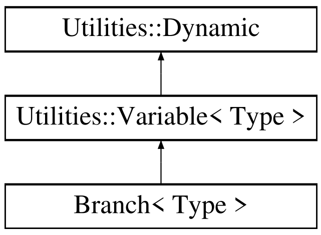
The documentation for this class was generated from the following file:

- /github/workspace/rapido/src/arbol.h

5.2 Branch< Type > Class Template Reference

```
#include <arbol.h>
```

Inheritance diagram for Branch< Type >:



Public Member Functions

- Branch ()
- Branch (TTree *ttree, TString new_branch_name)

Additional Inherited Members

5.2.1 Detailed Description

template<typename Type>
class Branch< Type >

Wraps TTree branches to allow for making branches on the fly

Template Parameters

Type	type of branch value

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Branch() [1/2]

```
template<typename Type >
Branch< Type >::Branch ( )
```

Branch object default constructor

Returns

none

5.2.2.2 Branch() [2/2]

```
template<typename Type >
Branch< Type >::Branch (
             TTree * ttree,
             TString new_branch_name )
```

Branch object constructor

Parameters

ttree	pointer to TTree
new_branch_name	new branch name

Returns

none

The documentation for this class was generated from the following file:

- /github/workspace/rapido/src/arbol.h

5.3 Utilities::CSVFile Class Reference

```
#include <utilities.h>
```

Public Member Functions

- CSVFile (std::ofstream &new_ofstream, std::string new_name, std::vector< std::string > new_headers)
- virtual ∼CSVFile ()
- CSVFile clone (std::string new_name)
- template<typename Type >
  void pushCol (Type value)
- void writeRow (bool append=true)

Public Attributes

- std::ofstream & ofstream
- std::string name
- std::vector< std::string > headers
- std::vector< std::string > buffer

5.3.1 Detailed Description

Object for handling CSV I/O

5.3.2 Constructor & Destructor Documentation

5.3.2.1 CSVFile()

```
Utilities::CSVFile::CSVFile (
            std::ofstream & new_ofstream,
            std::string new_name,
            std::vector< std::string > new_headers )
```

CSVFile object constructor

Parameters

new_ofstream	reference of an existing ofstream object
new_name	name of new CSV file (e.g. output.csv)
new_headers	headers for new CSV file columns

Returns

none

5.3.2.2 ∼CSVFile()

```
virtual Utilities::CSVFile::∼CSVFile ( )  [virtual]
```

CSVFile object destructor

Returns

none

5.3.3 Member Function Documentation

5.3.3.1 clone()

```
CSVFile Utilities::CSVFile::clone (
             std::string new_name )
```

Clone CSVFile object and copy the existing CSV file to a new file

Parameters

new_name	name of new CSV file (e.g. output.csv)

Returns

new CSVFile object

5.3.3.2 pushCol()

```
template<typename Type >
void Utilities::CSVFile::pushCol (
             Type value )
```

Push a new column entry to buffer

Template Parameters

Type	type of column entry

Parameters

value	new column entry

Returns

none

5.3.3.3 writeRow()

```
void Utilities::CSVFile::writeRow (
             bool append = true )
```

Write buffer to CSV file

Parameters

append	Toggle "append" mode (optional)

Returns

none

5.3.4 Member Data Documentation

5.3.4.1 buffer

```
std::vector<std::string> Utilities::CSVFile::buffer
```

Buffer for staging column values

5.3.4.2 headers

```
std::vector<std::string> Utilities::CSVFile::headers
```

Headers for CSV columns

5.3.4.3 name

```
std::string Utilities::CSVFile::name
```

Name (e.g. output.csv) of CSV file

5.3.4.4 ofstream

```
std::ofstream& Utilities::CSVFile::ofstream
```

fstream object for writing files

The documentation for this class was generated from the following file:

- /github/workspace/rapido/src/utilities.h

5.4 Cut Class Reference

```
#include <cutflow.h>
```

Public Member Functions

- Cut (std::string new_name, std::function< bool()> new_evaluate)
- Cut (std::string new_name, std::function< bool()> new_evaluate, std::function< float()> new_compute_↩ weight)
- void print (float weight=1.0)
- float getWeight ()

Public Attributes

- std::string name
- std::function< bool()> evaluate
- std::function< float()> compute_weight
- Cut ∗ parent
- Cut ∗ right
- Cut ∗ left
- int n_pass
- int n_fail
- float n_pass_weighted
- float n_fail_weighted

5.4.1 Detailed Description

Object that represents a single cut in an analysis

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Cut() [1/2]

```
Cut::Cut (
            std::string new_name,
            std::function< bool()> new_evaluate )
```

Cut object constructor (assumes weight == 1.0)

Parameters

new_name	new cut name
new_evaluate	lambda function that evaluates new cut conditional logic

Returns

none

5.4.2.2 Cut() [2/2]

```
Cut::Cut (
            std::string new_name,
            std::function< bool()> new_evaluate,
            std::function< float()> new_compute_weight )
```

Cut object constructor

Parameters

new_name	new cut name
new_evaluate	lambda function that evaluates new cut conditional logic
new_compute_weight	lambda function that computes event weight

Returns

none

5.4.3 Member Function Documentation

5.4.3.1 getWeight()

```
float Cut::getWeight ( )
```

Get even weight for this cut (on top of previous cut weights)

Returns

event weight

5.4.3.2 print()

```
void Cut::print (
            float weight = 1.0 )
```

Print cut object properties

Parameters

weight	event weight

Returns

none

5.4.4 Member Data Documentation

5.4.4.1 compute_weight

```
std::function<float()> Cut::compute_weight
```

Lambda function that computes event weight

5.4.4.2 evaluate

```
std::function<bool()> Cut::evaluate
```

Lambda function that evaluates conditional logic (i.e. the cut itself)

5.4.4.3 left

```
Cut* Cut::left
```

Pointer to next cut to evaluate if this cut evaluates to false

5.4.4.4 n_fail

```
int Cut::n_fail
```

Number of events that fail cut

5.4.4.5 n_fail_weighted

```
float Cut::n_fail_weighted
```

Weighted number of events that fail cut

5.4.4.6 n_pass

```
int Cut::n_pass
```

Number of events that pass cut

5.4.4.7 n_pass_weighted

```
float Cut::n_pass_weighted
```

Weighted number of events that pass cut

5.4.4.8 name

```
std::string Cut::name
```

Unique name of cut

5.4.4.9 parent

```
Cut* Cut::parent
```

Pointer to parent cut

5.4.4.10 right

```
Cut* Cut::right
```

Pointer to next cut to evaluate if this cut evaluates to true

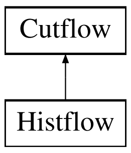
The documentation for this class was generated from the following files:

- /github/workspace/rapido/src/cutflow.h
- /github/workspace/rapido/src/cutflow.cc

5.5 Cutflow Class Reference

```
#include <cutflow.h>
```

Inheritance diagram for Cutflow:



Public Member Functions

- Cutflow ()
- Cutflow (std::string new_name)
- Cutflow (std::string new_name, Cut ∗new_root)
- ∼Cutflow ()
- void setRoot (Cut ∗new_root)
- void insert (std::string target_cut_name, Cut ∗new_cut, Direction direction)
- virtual bool run ()
- bool runUntil (std::string target_cut_name)
- Cut ∗ findTerminus (std::string starting_cut_name)
- void print ()
- void writeCSV (std::string output_dir="")

Public Attributes

- std::string name
- Utilities::Variables globals

Protected Member Functions

- Cut ∗ getCut (std::string cut_name)
- Cut ∗ recursiveFindTerminus (Cut ∗cut)
- void recursivePrint (std::string tabs, Cut ∗cut, Direction direction, float weight)
- std::pair< Cut ∗, bool > recursiveEvaluate (Cut ∗cut)
- void recursiveDelete (Cut ∗cut)

Protected Attributes

- Cut ∗ root
- std::map< std::string, Cut ∗ > cut_record

5.5.1 Detailed Description

An analysis represented as a binary search tree (i.e. analysis = tree, cut = node)

5.5.2 Constructor & Destructor Documentation

5.5.2.1 Cutflow() [1/3]

```
Cutflow::Cutflow ( )
```

Cutflow object default constructor

Returns

none

5.5.2.2 Cutflow() [2/3]

```
Cutflow::Cutflow (
              std::string new_name )
```

Cutflow object overload constructor

Parameters

new_name	name of cutflow

Returns

none

5.5.2.3 Cutflow() [3/3]

```
Cutflow::Cutflow (
            std::string new_name,
            Cut * new_root )
```

Cutflow object overload constructor

Parameters

new_name	name of cutflow
new_root	pointer to cut object to use as root node

Returns

none

5.5.2.4 ∼Cutflow()

```
Cutflow::∼Cutflow ( )
```

Cutflow object destructor

Returns

none

5.5.3 Member Function Documentation

5.5.3.1 findTerminus()

```
Cut * Cutflow::findTerminus (
            std::string starting_cut_name )
```

Find the rightmost terminal leaf from a given node

Parameters

starting_cut_name	cut from which to start search

Returns

terminal cut

5.5.3.2 getCut()

```
Cut * Cutflow::getCut (
             std::string cut_name )  [protected]
```

(PROTECTED) Retrieve cut object from cut record

Parameters

cut_name	cut name

Returns

pointer to cut

5.5.3.3 insert()

```
void Cutflow::insert (
             std::string target_cut_name,
             Cut * new_cut,
             Direction direction )
```

Insert a new node AFTER a given node

Parameters

target_cut_name	target node name
new_cut	pointer to new node
direction	direction (Left/false, Right/true)

Returns

none

5.5.3.4 print()

```
void Cutflow::print ( )
```

Print cutflow

Returns

none

5.5.3.5 recursiveDelete()

```
void Cutflow::recursiveDelete (
             Cut * cut )  [protected]
```

(PROTECTED) Recursively delete cuts in the cutflow

Parameters

cut	pointer to current cut

Returns

none

5.5.3.6 recursiveEvaluate()

```
std::pair< Cut *, bool > Cutflow::recursiveEvaluate (
             Cut * cut )  [protected]
```

(PROTECTED) Recursively evaulate cuts in the cutflow

Parameters

cut	pointer to current cut

Returns

std::pair of a pointer to terminal cut and a boolean (true = pass, false = fail)

5.5.3.7 recursiveFindTerminus()

```
Cut * Cutflow::recursiveFindTerminus (
             Cut * cut )  [protected]
```

(PROTECTED) Recursively search for the rightmost terminal leaf from a given node

Parameters

cut	pointer to current cut

Returns

terminal cut

5.5.3.8 recursivePrint()

```
void Cutflow::recursivePrint (
             std::string tabs,
             Cut * cut,
             Direction direction,
             float weight )  [protected]
```

(PROTECTED) Recursively print cuts

Parameters

tabs	string with the prefix tabs for current cut
cut	pointer to current cut
direction	direction of cut relative to parent
weight	current event weight

Returns

none

5.5.3.9 run()

```
bool Cutflow::run ( )  [virtual]
```

Run cutflow until any terminus

Returns

whether or not the terminal cut in the cutflow passed

Reimplemented in Histflow.

5.5.3.10 runUntil()

```
bool Cutflow::runUntil (
             std::string target_cut_name )
```

Run cutflow until a target terminal cut

See also

Cutflow::runUntil

Parameters

target_cut_name	name of target cut

Returns

whether or not (true/false) the target cut was reached and passed

5.5.3.11 setRoot()

```
void Cutflow::setRoot (
             Cut * new_root )
```

Set root node of cutflow object

Parameters

new_root	pointer to cut object to use as new root node

Returns

none

5.5.3.12 writeCSV()

```
void Cutflow::writeCSV (
             std::string output_dir = "" )
```

Print all cutflow paths to separate CSV files {output_dir}/{name}_{terminal_cut}.csv

Parameters

output_dir	target directory for output CSV files (optional)

Returns

none

5.5.4 Member Data Documentation

5.5.4.1 cut_record

`std::map<std::string, Cut*> Cutflow::cut_record [protected]`

Map ("record") of all cuts in cutflow

5.5.4.2 globals

`Utilities::Variables Cutflow::globals`

Dynamic list of variables to track across object scope (i.e. psuedo-members)

5.5.4.3 name

`std::string Cutflow::name`

Name of cutflow

5.5.4.4 root

`Cut* Cutflow::root [protected]`

Pointer to cut that is used as the root node

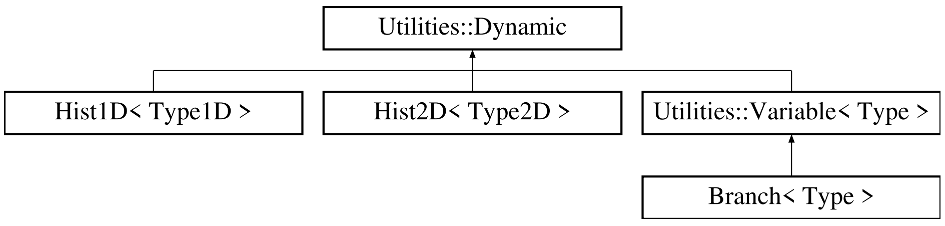
The documentation for this class was generated from the following files:

- /github/workspace/rapido/src/cutflow.h
- /github/workspace/rapido/src/cutflow.cc

5.6 Utilities::Dynamic Class Reference

`#include <utilities.h>`

Inheritance diagram for Utilities::Dynamic:



Public Member Functions

- virtual ∼Dynamic ()

5.6.1 Detailed Description

"Dynamic" object that serves as a base for templated objects

5.6.2 Constructor & Destructor Documentation

5.6.2.1 ∼Dynamic()

```
virtual Utilities::Dynamic::∼Dynamic ( ) [virtual]
```

Dynamic object destructor

Returns

none

The documentation for this class was generated from the following file:

- /github/workspace/rapido/src/utilities.h

5.7 HEPCLI Class Reference

```
#include <looper.h>
```

Public Member Functions

- HEPCLI ()
- HEPCLI (int argc, char ∗∗argv)

Public Attributes

- bool verbose
- std::string input_ttree
- std::string output_dir
- std::string output_name
- bool is_data
- bool is_signal
- float scale_factor
- TChain ∗ input_tchain

5.7.1 Detailed Description

Object for handling HEP CLI input (wraps getopt functionality)

5.7.2 Constructor & Destructor Documentation

5.7.2.1 HEPCLI() [1/2]

```
HEPCLI::HEPCLI ( )
```

HEPCLI object constructor

Returns

none

5.7.2.2 HEPCLI() [2/2]

```
HEPCLI::HEPCLI (
              int argc,
              char ** argv )
```

HEPCLI object overload constructor

Parameters

argc	argument count
argv	argument vector

Returns

none

5.7.3 Member Data Documentation

5.7.3.1 input_tchain

```
TChain* HEPCLI::input_tchain
```

ROOT TChain with input files

5.7.3.2 input_ttree

```
std::string HEPCLI::input_ttree
```

Name of TTree in input ROOT file(s)

5.7.3.3 is_data

```
bool HEPCLI::is_data
```

Data (as opposed to Monte Carlo) flag

5.7.3.4 is_signal

```
bool HEPCLI::is_signal
```

Signal (as opposed to background) flag

5.7.3.5 output_dir

```
std::string HEPCLI::output_dir
```

Target directory for output file(s)

5.7.3.6 output_name

```
std::string HEPCLI::output_name
```

Short name for output file(s)

5.7.3.7 scale_factor

```
float HEPCLI::scale_factor
```

Global event weight

5.7.3.8 verbose

```
bool HEPCLI::verbose
```

Verbosity flag

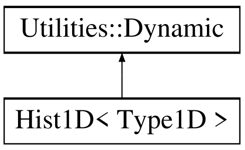
The documentation for this class was generated from the following file:

- /github/workspace/rapido/src/looper.h

5.8 Hist1D< Type1D > Class Template Reference

```
#include <histflow.h>
```

Inheritance diagram for Hist1D< Type1D >:



Public Member Functions

- Hist1D (Type1D ∗new_hist, Filler1D new_filler)
- ∼Hist1D ()
- void fill (float weight=1.0)
- void write ()
- Hist1D< Type1D > ∗ clone ()

Public Attributes

- TString name

5.8.1 Detailed Description

**template<typename Type1D>
class Hist1D< Type1D >**

"Dynamic" 1D ROOT histogram object

Template Parameters

Type1D	type of 1D ROOT histogram (e.g. TH1F)

5.8.2 Constructor & Destructor Documentation

5.8.2.1 Hist1D()

```
template<typename Type1D >
Hist1D< Type1D >::Hist1D (
             Type1D ∗ new_hist,
             Filler1D new_filler )
```

1D Histogram constructor

Parameters

new_hist	pointer to a 1D ROOT histogram
new_filler	lambda function that computes the value used to fill the histogram

Returns

none

5.8.2.2 ∼Hist1D()

```
template<typename Type1D >
Hist1D< Type1D >::∼Hist1D ( )
```

1D Histogram destructor

Returns

none

5.8.3 Member Function Documentation

5.8.3.1 clone()

```
template<typename Type1D >
Hist1D< Type1D > * Hist1D< Type1D >::clone ( )
```

Clone this "dynamic" histogram object

Returns

none

5.8.3.2 fill()

```
template<typename Type1D >
void Hist1D< Type1D >::fill (
              float weight = 1.0 )
```

Call filler to fill histogram with an optional weight

Parameters

weight	float to weigh new histogram entry (optional)

Returns

none

5.8.3.3 write()

```
template<typename Type1D >
void Hist1D< Type1D >::write ( )
```

Write ROOT histogram to currently opened TFile

Returns

none

5.8.4 Member Data Documentation

5.8.4.1 name

```
template<typename Type1D >
TString Hist1D< Type1D >::name
```

Name of histogram

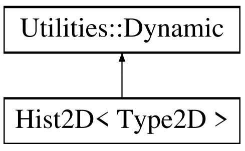
The documentation for this class was generated from the following file:

- /github/workspace/rapido/src/histflow.h

5.9 Hist2D< Type2D > Class Template Reference

```
#include <histflow.h>
```

Inheritance diagram for Hist2D< Type2D >:



Public Member Functions

- Hist2D (Type2D ∗new_hist, Filler2D new_filler)
- ∼Hist2D ()
- void fill (float weight=1.0)
- void write ()
- Hist2D ∗ clone ()

Public Attributes

- TString name

5.9.1 Detailed Description

template<typename Type2D>
class Hist2D< Type2D >

"Dynamic" 2D ROOT histogram object

Template Parameters

Type2D	type of 2D ROOT histogram (e.g. TH2F)

5.9.2 Constructor & Destructor Documentation

5.9.2.1 Hist2D()

```
template<typename Type2D >
Hist2D< Type2D >::Hist2D (
             Type2D ∗ new_hist,
             Filler2D new_filler )
```

2D Histogram constructor

Parameters

new_hist	pointer to a 2D ROOT histogram
new_filler	lambda function that computes the value used to fill the histogram

Returns

none

5.9.2.2 ∼Hist2D()

```
template<typename Type2D >
Hist2D< Type2D >::∼Hist2D ( )
```

2D Histogram destructor

Returns

none

5.9.3 Member Function Documentation

5.9.3.1 clone()

```
template<typename Type2D >
Hist2D ∗ Hist2D< Type2D >::clone ( )
```

Clone this "dynamic" histogram object

Returns

none

5.9.3.2 fill()

```
template<typename Type2D >
void Hist2D< Type2D >::fill (
              float weight = 1.0 )
```

Call filler to fill histogram with an optional weight

Parameters

weight	float to weigh new histogram entry (optional)

Returns

none

5.9.3.3 write()

```
template<typename Type2D >
void Hist2D< Type2D >::write ( )
```

Write ROOT histogram to currently opened TFile

Returns

none

5.9.4 Member Data Documentation

5.9.4.1 name

```
template<typename Type2D >
TString Hist2D< Type2D >::name
```

Name of histogram

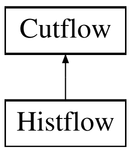
The documentation for this class was generated from the following file:

- /github/workspace/rapido/src/histflow.h

5.10 Histflow Class Reference

```
#include <histflow.h>
```

Inheritance diagram for Histflow:



Public Member Functions

- Histflow ()
- ∼Histflow ()
- template<typename Type1D >
  void bookHist1D (std::string target_cut_name, Hist1D< Type1D > ∗hist)
- template<typename Type2D >
  void bookHist2D (std::string target_cut_name, Hist2D< Type2D > ∗hist)
- template<typename Type1D >
  void bookHist1D (std::string target_cut_name, Type1D ∗hist, Filler1D filler)
- template<typename Type2D >
  void bookHist2D (std::string target_cut_name, Type2D ∗hist, Filler2D filler)
- void writeHists (TFile ∗tfile)
- Cut ∗ run () override

Protected Member Functions

- Cut ∗ recursiveEvaluate (Cut ∗cut, float weight=1.0)

Protected Attributes

- std::map< std::string, std::vector< std::function< void(float)> > > fill_schedule
- std::map< TString, std::function< void()> > hist_writers

Additional Inherited Members

5.10.1 Detailed Description

Modified Cutflow object that fills booked histograms after passing a given set of cuts

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Histflow()

```
Histflow::Histflow ( )
```

Histflow constructor

Returns

none

5.10.2.2 ∼Histflow()

```
Histflow::∼Histflow ( )
```

Histflow destructor

Returns

none

5.10.3 Member Function Documentation

5.10.3.1 bookHist1D() [1/2]

```
template<typename Type1D >
void Histflow::bookHist1D (
             std::string target_cut_name,
             Hist1D< Type1D > ∗ hist )
```

Schedule a "dynamic" 1D histogram object for a given cut

Parameters

target_cut_name	target node name
hist	pointer to "dynamic" 1D histogram object to schedule

Returns

none

5.10.3.2 bookHist1D() [2/2]

```
template<typename Type1D >
void Histflow::bookHist1D (
             std::string target_cut_name,
             Type1D * hist,
             Filler1D filler )
```

Schedule a 1D ROOT histogram for a given cut

Parameters

target_cut_name	target node name
hist	pointer to 1D ROOT histogram to schedule
filler	lambda function that computes the value used to fill the histogram

Returns

none

5.10.3.3 bookHist2D() [1/2]

```
template<typename Type2D >
void Histflow::bookHist2D (
             std::string target_cut_name,
             Hist2D< Type2D > * hist )
```

Schedule a "dynamic" 2D histogram object for a given cut

Parameters

target_cut_name	target node name
hist	pointer to "dynamic" 2D histogram object to schedule

Returns

none

5.10.3.4 bookHist2D() [2/2]

```
template<typename Type2D >
void Histflow::bookHist2D (
             std::string target_cut_name,
             Type2D * hist,
             Filler2D filler )
```

Schedule a 2D ROOT histogram for a given cut

Parameters

target_cut_name	target node name
hist	pointer to 2D ROOT histogram to schedule
filler	lambda function that computes the value used to fill the histogram

Returns

none

5.10.3.5 recursiveEvaluate()

```
Cut * Histflow::recursiveEvaluate (
             Cut * cut,
             float weight = 1.0 )  [protected]
```

(PROTECTED) Additional definition that recursively evaluates cuts in cutflow and fills scheduled histograms when appropriate cuts are passed

Parameters

cut	pointer to current cut
weight	current event weight (optional)

Returns

none

5.10.3.6 run()

```
Cut * Histflow::run ( )  [override], [virtual]
```

Overriding definition that runs cutflow with Histflow::recursiveEvaluate

Returns

pointer to terminal cut (final leaf of tree reached)

Reimplemented from Cutflow.

5.10.3.7 writeHists()

```
void Histflow::writeHists (
             TFile * tfile )
```

Write all histograms to a given TFile

Parameters

tfile	pointer to ROOT TFile to write histograms to

Returns

none

5.10.4 Member Data Documentation

5.10.4.1 fill_schedule

```
std::map<std::string, std::vector<std::function<void(float)> > > Histflow::fill_schedule
[protected]
```

"Schedule" dictating when to fill certain histograms

5.10.4.2 hist_writers

```
std::map<TString, std::function<void()> > Histflow::hist_writers  [protected]
```

Collection of functions that write histograms to opened TFile

The documentation for this class was generated from the following file:

- /github/workspace/rapido/src/histflow.h

5.11 Looper Class Reference

```
#include <looper.h>
```

Public Member Functions

- Looper (TChain ∗new_tchain)
- Looper (TChain ∗new_tchain, TString new_ttree_name)
- virtual ∼Looper ()
- void run (std::function< void(TTree ∗ttree)> init, std::function< void(int entry)> eval)

Public Attributes

- unsigned int current_entry
- unsigned int n_events_processed
- unsigned int n_events_to_process

5.11.1 Detailed Description

Object to handle looping over ROOT files

5.11.2 Constructor & Destructor Documentation

5.11.2.1 Looper() [1/2]

```
Looper::Looper (
            TChain ∗ new_tchain )
```

Looper object constructor

Parameters

new_tchain	pointer to ROOT TChain of files to loop over

Returns

none

5.11.2.2 Looper() [2/2]

```
Looper::Looper (
            TChain ∗ new_tchain,
            TString new_ttree_name )
```

Looper object overload constructor

Parameters

new_tchain	pointer to ROOT TChain of files to loop over
new_ttree_name	name of the ROOT TTree

Returns

none

5.11.2.3 ∼Looper()

```
virtual Looper::∼Looper ( ) [virtual]
```

Looper object destructor

Returns

none

5.11.3 Member Function Documentation

5.11.3.1 run()

```
void Looper::run (
            std::function< void(TTree *ttree)> init,
            std::function< void(int entry)> eval )
```

Run looper with file- and event-processing logic captured in void lambda functions.

The following example uses a class named "Selector" generated by ROOT::MakeSelector; this class requires certain file- and event-processing initialization steps:

```
int main()
{
    TChain* tchain = new TChain("Events");
    tchain->Add("/path/to/file.root");
    selector = Selector(); // generated by ROOT::MakeSelector
    looper = Looper(tchain, "Events");
    looper.run(
        [&](TTree* ttree) { selector.Init(ttree); },
        [&](int entry)
        {
            selector.GetEntry(entry);
            selector.Process(entry);
            // -> insert your favorite cutflow here <--
        }
    );
}
```

Parameters

init	file-level initialization steps captured in a void lambda function
eval	event-level logic captured in a void lambda function

Returns

none

5.11.4 Member Data Documentation

5.11.4.1 current_entry

```
unsigned int Looper::current_entry
```

Current entry in TTree (i.e. current index of event loop)

5.11.4.2 n_events_processed

```
unsigned int Looper::n_events_processed
```

Number of events that have been processed

5.11.4.3 n_events_to_process

```
unsigned int Looper::n_events_to_process
```

Number of events in the TChain

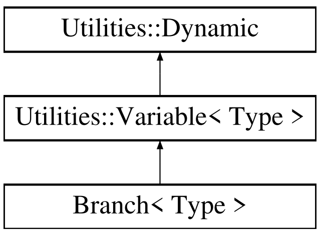
The documentation for this class was generated from the following file:

- /github/workspace/rapido/src/looper.h

5.12 Utilities::Variable< Type > Class Template Reference

```
#include <utilities.h>
```

Inheritance diagram for Utilities::Variable< Type >:



Public Member Functions

- Variable ()
- Variable (Type new_reset_value)
- virtual ∼Variable ()
- Type getValue ()
- Type & getReference ()
- void setValue (Type new_value)
- void setResetValue (Type new_reset_value)
- void resetValue ()

Protected Attributes

- Type value
- Type reset_value

5.12.1 Detailed Description

template<typename Type>
class Utilities::Variable< Type >

"Dynamic" variable

Template Parameters

Type	type of variable

5.12.2 Constructor & Destructor Documentation

5.12.2.1 Variable() [1/2]

```
template<typename Type >
Utilities::Variable< Type >::Variable ( )
```

Variable object default constructor

Returns

none

5.12.2.2 Variable() [2/2]

```
template<typename Type >
Utilities::Variable< Type >::Variable (
              Type new_reset_value )
```

Variable object overload constructor

Parameters

new_reset_value	reset value of new variable object

Returns

none

5.12.2.3 ∼Variable()

```
template<typename Type >
virtual Utilities::Variable< Type >::∼Variable ( )  [virtual]
```

Variable object destructor

Returns

none

5.12.3 Member Function Documentation

5.12.3.1 getReference()

```
template<typename Type >
Type & Utilities::Variable< Type >::getReference ( )
```

Get reference to variable value

Returns

reference to value for this variable object

5.12.3.2 getValue()

```
template<typename Type >
Type Utilities::Variable< Type >::getValue ( )
```

Get current variable value

Returns

current value of this variable object

5.12.3.3 resetValue()

```
template<typename Type >
void Utilities::Variable< Type >::resetValue ( )
```

Reset the current variable value to the reset value

Returns

none

5.12.3.4 setResetValue()

```
template<typename Type >
void Utilities::Variable< Type >::setResetValue (
             Type new_reset_value )
```

Set variable reset value

Parameters

new_reset_value	new reset value (e.g. -999; default is the default type constructor)

Returns

none

5.12.3.5 setValue()

```
template<typename Type >
void Utilities::Variable< Type >::setValue (
             Type new_value )
```

Set variable value

Parameters

new_value	new value

Returns

none

5.12.4 Member Data Documentation

5.12.4.1 reset_value

```
template<typename Type >
Type Utilities::Variable< Type >::reset_value  [protected]
```

Variable reset value

5.12.4.2 value

```
template<typename Type >
Type Utilities::Variable< Type >::value  [protected]
```

Variable value

The documentation for this class was generated from the following file:

- /github/workspace/rapido/src/utilities.h

5.13 Utilities::Variables Class Reference

```
#include <utilities.h>
```

Public Member Functions

- Variables ()
- virtual ~Variables ()
- template<typename Type >
  void newVar (std::string new_name)
- template<typename Type >
  void newVar (std::string new_name, Type new_reset_value)
- template<typename Type >
  Type getVal (std::string name)
- template<typename Type >
  Type & getRef (std::string name)
- template<typename Type >
  void setVal (std::string name, Type new_value)
- template<typename Type >
  void resetVal (std::string name)
- void resetVars ()

Protected Member Functions

- template<typename Type >
  Variable< Type > * getVar (std::string name)

Protected Attributes

- std::map< std::string, Dynamic * > variables
- std::map< std::string, std::function< void()> > resetters

5.13.1 Detailed Description

A group of "dynamic" variables

5.13.2 Constructor & Destructor Documentation

5.13.2.1 Variables()

```
Utilities::Variables::Variables ( )
```

Variables object constructor

Returns

none

5.13.2.2 ∼Variables()

```
virtual Utilities::Variables::∼Variables ( )  [virtual]
```

Variables object destructor

Returns

none

5.13.3 Member Function Documentation

5.13.3.1 getRef()

```
template<typename Type >
Type & Utilities::Variables::getRef (
            std::string name )
```

Get variable value in map by reference if it exists

Template Parameters

Type	type of variable

Parameters

name	name of variable

Returns

none

5.13.3.2 getVal()

```
template<typename Type >
Type Utilities::Variables::getVal (
            std::string name )
```

Get variable value in map if it exists

Template Parameters

Type	type of variable

Parameters

name	name of variable

Returns

none

5.13.3.3 getVar()

```
template<typename Type >
Variable< Type > * Utilities::Variables::getVar (
            std::string name ) [protected]
```

(PROTECTED) Retrieve variable object from map if it exists

Template Parameters

Type	type of variable

Parameters

name	name of variable

Returns

none

5.13.3.4 newVar() [1/2]

```
template<typename Type >
void Utilities::Variables::newVar (
             std::string new_name )
```

Add blank variable to map

Template Parameters

Type	type of new variable

Parameters

new_name	name of new variable

Returns

none

5.13.3.5 newVar() [2/2]

```
template<typename Type >
void Utilities::Variables::newVar (
             std::string new_name,
             Type new_reset_value )
```

Add new variable to map with reset value

Template Parameters

Type	type of variable

Parameters

new_name	name of variable
new_reset_value	reset value of new variable

Returns

none

5.13.3.6 resetVal()

```
template<typename Type >
void Utilities::Variables::resetVal (
             std::string name )
```

Set value of a variable in map to its reset value if it exists

Template Parameters

Type	type of variable

Parameters

name	name of variable

Returns

none

5.13.3.7 resetVars()

```
void Utilities::Variables::resetVars ( )
```

Set value of each variable in map to its respective reset value.

Uses a map of "resetters" because Utilities::Variable<Type>::resetValue() cannot be called across an arbitrary number of such objects, due to the fact that the value of Type for each object would need to be supplied. The "resetters" circumvent this issue by capturing the function call in a lambda function for later use.

Returns

none

5.13.3.8 setVal()

```
template<typename Type >
void Utilities::Variables::setVal (
             std::string name,
             Type new_value )
```

Set value of a variable in map if it exists

Template Parameters

Type	type of variable

Parameters

name	name of variable
new_value	new value for variable

Returns

none

5.13.4 Member Data Documentation

5.13.4.1 resetters

```
std::map<std::string, std::function<void()> > Utilities::Variables::resetters  [protected]
```

Map of Utilities::Variable::resetValue functions captured in lambdas

5.13.4.2 variables

```
std::map<std::string, Dynamic*> Utilities::Variables::variables  [protected]
```

Map of Utilities::Variable objects

The documentation for this class was generated from the following file:

- /github/workspace/rapido/src/utilities.h

Chapter 6

File Documentation

6.1 arbol.h

```cpp
#ifndef ARBOL_H
#define ARBOL_H

#include <iostream>
#include <functional>
#include <string>
#include <vector>
#include <map>

#include "TString.h"
#include "TTree.h"
#include "TFile.h"

#include "utilities.h"

template<typename Type>
class Branch : public Utilities::Variable<Type>
{
private:
    TBranch* branch;
public:
    Branch();
    Branch(TTree* ttree, TString new_branch_name);
};

class Arbol
{
protected:
    std::map<TString, Utilities::Dynamic*> branches;
    std::map<TString, std::function<void()» branch_resetters;
    template<typename Type>
    Branch<Type>* getBranch(TString branch_name);
public:
    TTree* ttree;
    TFile* tfile;

    Arbol();
    Arbol(TFile* new_tfile);
    virtual ~Arbol();
    template<typename Type>
    void newBranch(TString new_branch_name);
    template<typename Type>
    void newBranch(TString new_branch_name, Type new_reset_value);
    template<typename Type>
    void setBranchResetValue(TString branch_name, Type new_reset_value);
    template<typename Type>
    Type getLeaf(TString branch_name);
    template<typename Type>
    void setLeaf(TString branch_name, Type new_value);

    template<typename Type>
    void newVecBranch(TString new_branch_name);
    template<typename Type>
    void newVecBranch(TString new_branch_name, std::vector<Type> new_reset_vector);
    template<typename Type>
    void setVecBranchResetValue(TString branch_name, std::vector<Type> new_reset_vector);
    template<typename Type>
    std::vector<Type> getVecLeaf(TString branch_name);
    template<typename Type>
    void setVecLeaf(TString branch_name, std::vector<Type> new_vector);

    template<typename Type>
    void appendToVecLeaf(TString branch_name, Type new_value);
    template<typename Type>
    void prependToVecLeaf(TString branch_name, Type new_value);
    template<typename Type>
    void insertIntoVecLeaf(TString branch_name, Type new_value, int index);
    template<typename Type>
    void sortVecLeaf(TString branch_name, std::function<bool(Type, Type)> &lambda);

    void resetBranches();

    void fillTTree();
    void writeTFile();
};

#include "arbol.icc"

#endif
```

6.2 cutflow.h

```
#ifndef CUTFLOW_H
#define CUTFLOW_H

#include <fstream>
#include <iostream>
#include <functional>
#include <string>
#include <vector>
#include <map>

#include "utilities.h"

enum Direction
{
    Left,
    Right
};
typedef std::vector<Direction> Directions;

class Cut
{
public:
    std::string name;
    std::function<bool()> evaluate;
    std::function<float()> compute_weight;
    Cut* parent;
    Cut* right;
    Cut* left;
    int n_pass;
    int n_fail;
    float n_pass_weighted;
    float n_fail_weighted;

    Cut(std::string new_name, std::function<bool()> new_evaluate);
    Cut(std::string new_name, std::function<bool()> new_evaluate,
        std::function<float()> new_compute_weight);
    void print(float weight = 1.0);
    float getWeight();
};

class Cutflow
{
private:
    void recursiveWrite(std::string output_dir, Cut* cut, Direction direction, int csv_idx,
                        Utilities::CSVFiles csv_files, float weight);
protected:
    Cut* root;
    std::map<std::string, Cut*> cut_record;
     Cut* getCut(std::string cut_name);
     Cut* recursiveFindTerminus(Cut* cut);
     void recursivePrint(std::string tabs, Cut* cut, Direction direction, float weight);
     std::pair<Cut*, bool> recursiveEvaluate(Cut* cut);
     void recursiveDelete(Cut* cut);
public:
     std::string name;
     Utilities::Variables globals;

     Cutflow();
     Cutflow(std::string new_name);
    Cutflow(std::string new_name, Cut* new_root);
    ~Cutflow();
    void setRoot(Cut* new_root);
    void insert(std::string target_cut_name, Cut* new_cut, Direction direction);
    virtual bool run();
    bool runUntil(std::string target_cut_name);
    Cut* findTerminus(std::string starting_cut_name);
    void print();
    void writeCSV(std::string output_dir = "");
};

#endif
```

6.3 histflow.h

```
#ifndef HISTOS_H
#define HISTOS_H

#include <functional>
#include <map>

#include "cutflow.h"
#include "utilities.h"

typedef std::function<float()> Filler1D;
typedef std::function<pair<float, float>()> Filler2D;

template<typename Type1D>
class Hist1D : public Utilities::Dynamic
{
private:
    Type1D* hist;
    Filler1D filler;
public:
    TString name;

    Hist1D(Type1D* new_hist, Filler1D new_filler);
    ~Hist1D();
    void fill(float weight = 1.0);
    void write();
    Hist1D<Type1D>* clone();
};

template<typename Type2D>
class Hist2D : public Utilities::Dynamic
{
private:
    Type2D* hist;
    Filler2D filler;
public:
    TString name;

    Hist2D(Type2D* new_hist, Filler2D new_filler);
    ~Hist2D();
    void fill(float weight = 1.0);
    void write();
    Hist2D* clone();
};

class Histflow : public Cutflow
{
protected:
    std::map<std::string, std::vector<std::function<void(float)>»> fill_schedule;
    std::map<TString, std::function<void()» hist_writers;
    Cut* recursiveEvaluate(Cut* cut, float weight = 1.0);
public:
    Histflow();
    ~Histflow();
    template<typename Type1D>
    void bookHist1D(std::string target_cut_name, Hist1D<Type1D>* hist);
    template<typename Type2D>
    void bookHist2D(std::string target_cut_name, Hist2D<Type2D>* hist);
    template<typename Type1D>
    void bookHist1D(std::string target_cut_name, Type1D* hist, Filler1D filler);
    template<typename Type2D>
    void bookHist2D(std::string target_cut_name, Type2D* hist, Filler2D filler);
    void writeHists(TFile* tfile);
    Cut* run() override;
};

#include "histflow.icc"

#endif
```

6.4 looper.h

```
#ifndef LOOPER_H
#define LOOPER_H

#include <functional>
#include <iostream>
#include <iomanip>
#include <string>
#include <stdlib.h>
#include <getopt.h>

#include "TString.h"
#include "TChain.h"
#include "TFile.h"
#include "TTree.h"
#include "TTreeCache.h"
#include "TTreeCacheUnzip.h"

class HEPCLI
{
private:
    void printHelp();
    void parse(int argc, char** argv);
public:
    bool verbose;
    std::string input_ttree;
    std::string output_dir;
    std::string output_name;
    bool is_data;
    bool is_signal;
    float scale_factor;
    TChain* input_tchain;

    HEPCLI();

    HEPCLI(int argc, char** argv);
};

class Looper
{
private:
    TChain* tchain;
    TString ttree_name;
public:
    unsigned int current_entry;
    unsigned int n_events_processed;
    unsigned int n_events_to_process;

    Looper(TChain* new_tchain);
    Looper(TChain* new_tchain, TString new_ttree_name);
     virtual ~Looper();
     void run(std::function<void(TTree* ttree)> init, std::function<void(int entry)> eval);
 };

 #include "looper.icc"

 #endif
```

6.5 utilities.h

```
#ifndef UTILITIES_H
#define UTILITIES_H

#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <map>

namespace Utilities
{
    class CSVFile
    {
    public:
        std::ofstream& ofstream;
        std::string name;
        std::vector<std::string> headers;
        std::vector<std::string> buffer;

        CSVFile(std::ofstream& new_ofstream, std::string new_name,
                std::vector<std::string> new_headers);
        virtual ~CSVFile();
        CSVFile clone(std::string new_name);
        template<typename Type>
        void pushCol(Type value);
        void writeRow(bool append = true);
    };
    typedef std::vector<CSVFile> CSVFiles;

    class Dynamic
    {
    public:
        virtual ~Dynamic();
    };

    template<typename Type>
    class Variable : public Dynamic
    {
    protected:
        Type value;
        Type reset_value;
    public:
        Variable();
        Variable(Type new_reset_value);
        virtual ~Variable();
        Type getValue();
        Type& getReference();
        void setValue(Type new_value);
        void setResetValue(Type new_reset_value);
        void resetValue();
    };

    class Variables
    {
    protected:
        std::map<std::string, Dynamic*> variables;
        std::map<std::string, std::function<void()» resetters;
        template<typename Type>
        Variable<Type>* getVar(std::string name);
    public:
        Variables();
        virtual ~Variables();
        template<typename Type>
        void newVar(std::string new_name);
        template<typename Type>
        void newVar(std::string new_name, Type new_reset_value);
        template<typename Type>
        Type getVal(std::string name);
        template<typename Type>
        Type& getRef(std::string name);
        template<typename Type>
        void setVal(std::string name, Type new_value);
        template<typename Type>
        void resetVal(std::string name);
        void resetVars();
    };
}

#include "utilities.icc"

#endif
```

Index