New Features for HistFactory

George Lewis

Kyle Cranmer

New Features

Currently, using Histfactory requires writing a model in xml, which points to already-made histograms stored in ROOT files

These xml files are then parsed and a model is created using the ‘hist2workspace’ command-line executable

This setup has been used by many groups, including Higgs, Top, SUSY and this style of using HistFactory will still be available

The new features that we’ve developed will allow for additional means of interacting with Histfactory:

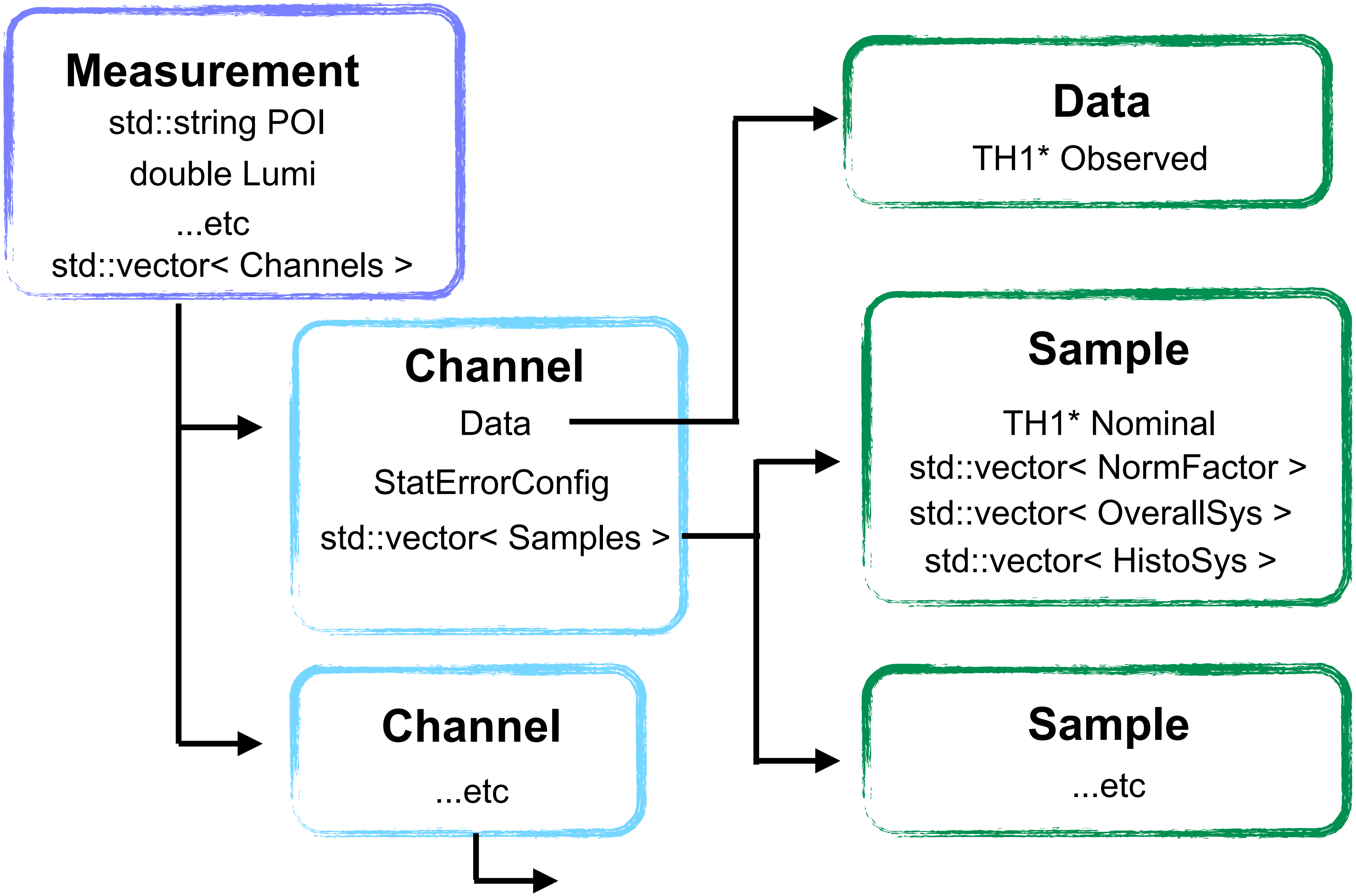
- Ability to create models directly within c++ or python
- Automatic saving of model structure, input histograms
- Ability to remake input xml files from output ROOT file

Updates to c++ structure

The configuration part of Histfactory now is based on a c++ class structure. These classes include:

- These classes mirror the structure of the input xml tags
- The xml parser now simply walks the xml DOM structure, creates these c++ classes, and feeds them into the modeling/fitting part of the code
- These c++ are classes automatically saved in the output ROOT file, along with the Workspace
- They can be browsed via the ROOT command line or can be imported into a script

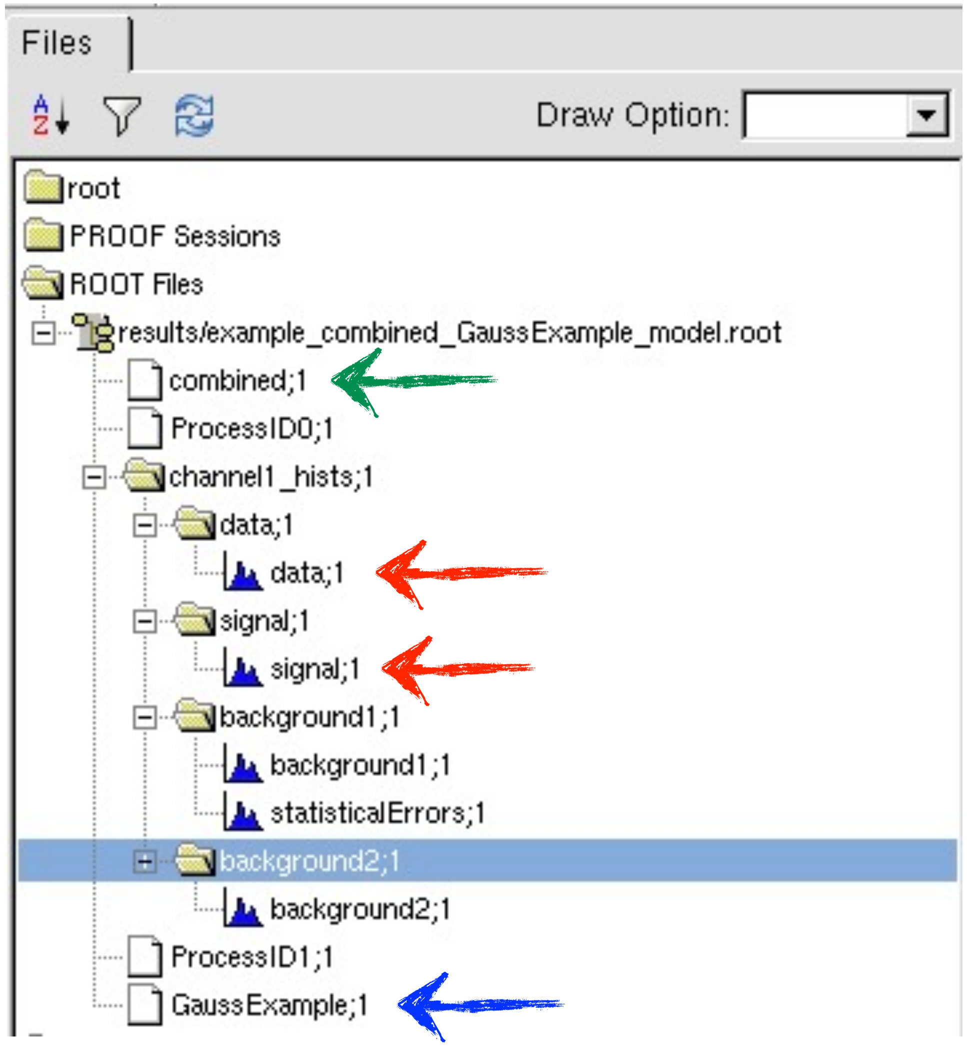
Class Tree Structure (schematic)



hist2workspace

- The command hist2workspace now saves all input histograms in the output ROOT file
- The measurement's configuration class is persistifed as well (here it is "GaussExample")
- The measurement has a method which can write its own xml files which point to the histograms saved in this root file:

GaussExample->WriteToXML()



```
Files
Draw Option:
root
PROOF Sessions
ROOT Files
results/example_combined_GaussExample_model.root
  combined;1
  ProcessID0;1
  channel1_hists;1
    data;1
      data;1
    signal;1
      signal;1
    background1;1
      background1;1
      statisticalErrors;1
    background2;1
      background2;1
  ProcessID1;1
  GaussExample;1
```

Scripting

Since the configuration state of HistFactory is now based on c++, one can create a histfactory model in various new ways:

- Using a simple ROOT script / compiled c++
- Using Python and PyROOT
- By editing a preexisting model created with XML

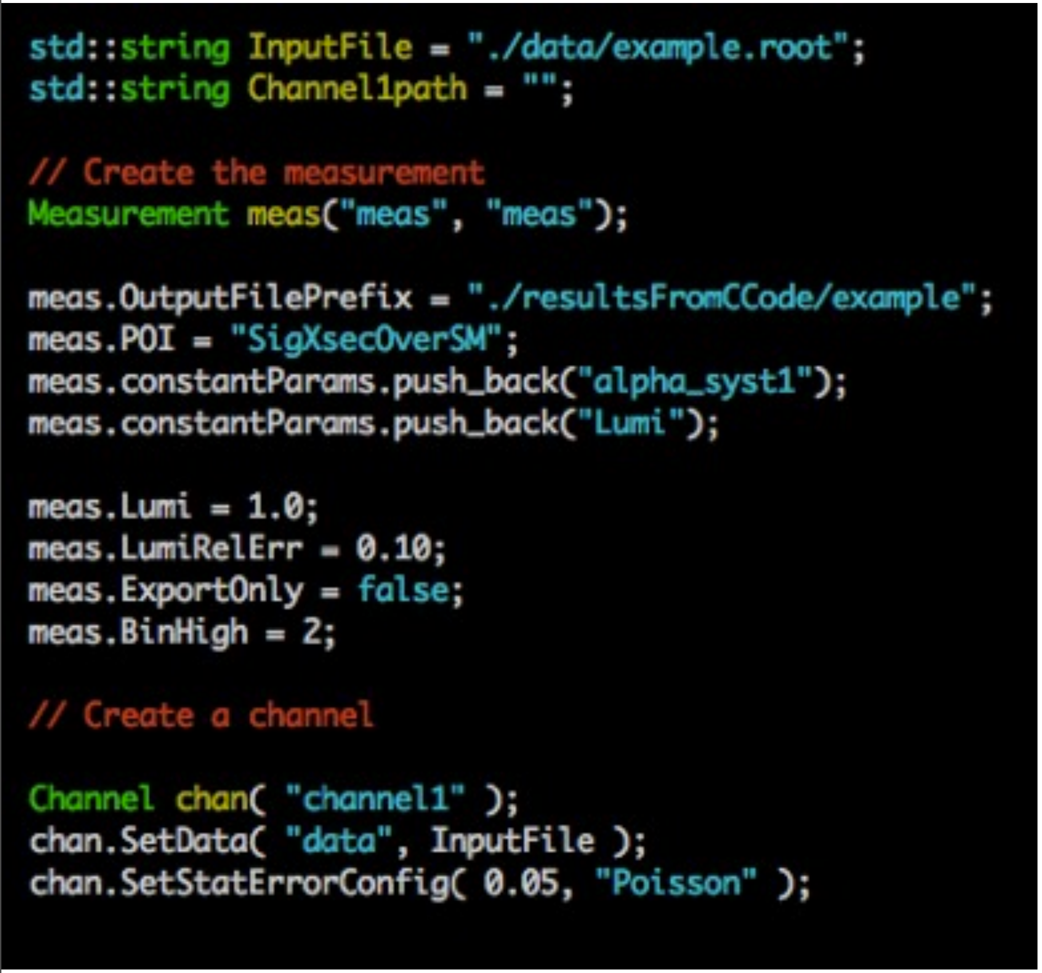
One can therefore bypass XML if one chooses (all of the new classes have python bindings and dictionaries).

All functions used in ‘hist2workspace’ are available for a script

Alternatively, one can create a model in xml, load the configuration structure in a script, modify the configuration, and create an altered version of the model

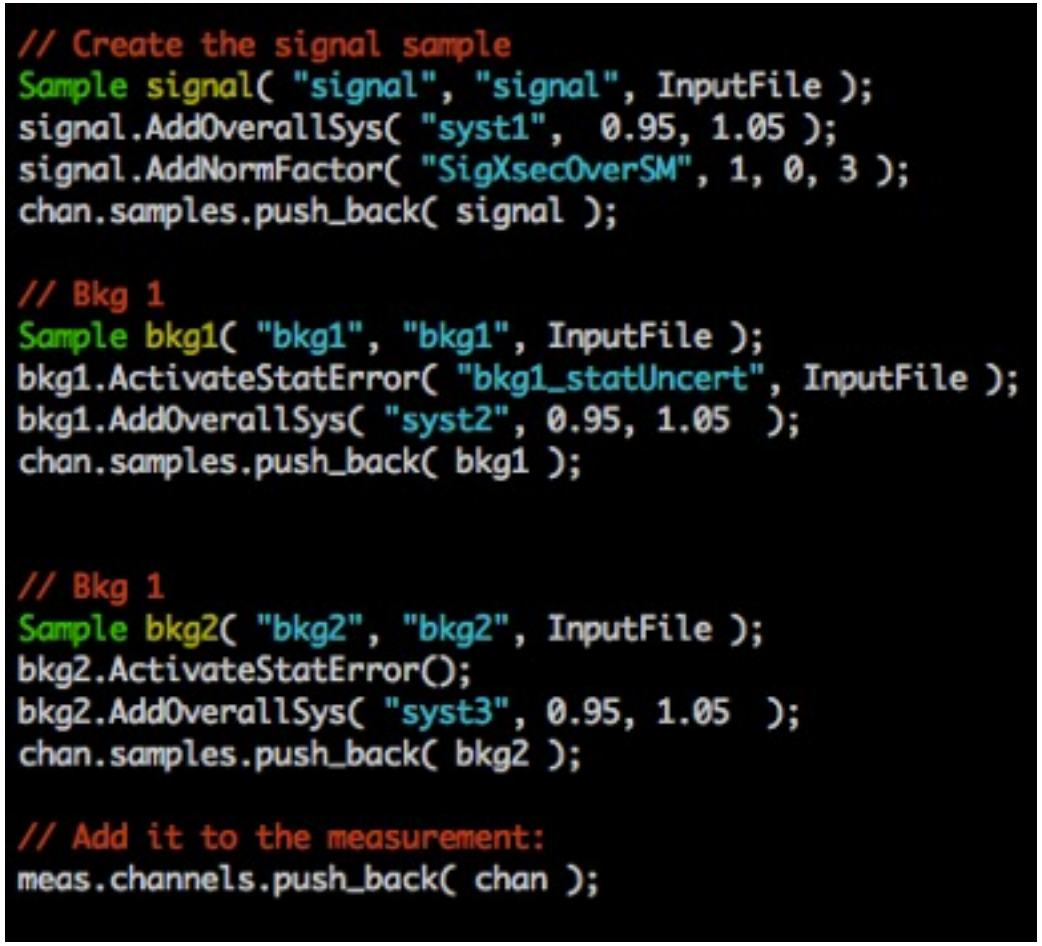
Example of Scripting (c++)

Create the Measurement and a channel



```
std::string InputFile = "./data/example.root";
std::string Channel1path = "";

// Create the measurement
Measurement meas("meas", "meas");

meas.OutputFilePrefix = "./resultsFromCCode/example";
meas.POI = "SigXsecOverSM";
meas.constantParams.push_back("alpha_syst1");
meas.constantParams.push_back("Lumi");

meas.Lumi = 1.0;
meas.LumiRelErr = 0.10;
meas.ExportOnly = false;
meas.BinHigh = 2;

// Create a channel

Channel chan( "channel1" );
chan.SetData( "data", InputFile );
chan.SetStatErrorConfig( 0.05, "Poisson" );
```

Create the channel's samples, including signal and background



```
// Create the signal sample
Sample signal( "signal", "signal", InputFile );
signal.AddOverallSys( "syst1",  0.95, 1.05 );
signal.AddNormFactor( "SigXsecOverSM", 1, 0, 3 );
chan.samples.push_back( signal );

// Bkg 1
Sample bkg1( "bkg1", "bkg1", InputFile );
bkg1.ActivateStatError( "bkg1_statUncert", InputFile );
bkg1.AddOverallSys( "syst2", 0.95, 1.05  );
chan.samples.push_back( bkg1 );


// Bkg 1
Sample bkg2( "bkg2", "bkg2", InputFile );
bkg2.ActivateStatError();
bkg2.AddOverallSys( "syst3", 0.95, 1.05  );
chan.samples.push_back( bkg2 );

// Add it to the measurement:
meas.channels.push_back( chan );
```

Advantages of new features

- Persistifying the configuration and the ability to recreate the xml structure are quite helpful in saving models, passing them between analyzers, and having experts debug them
- Scripting of models allows one to easily create many similar models based on an initial template (for example, correlating various parameters, changing sizes of systematics, switching input histograms, etc). Previously, one had to create many similar xml files and run hist2workspace many times.
- Saving input histograms is also helpful for debugging, and for making plots comparing the input histograms to the final fitted shapes
- In addition, we have introduced some stricter checks in inputs

Conclusions

We’re now doing some final stylistic changes to the code (to conform to ROOT conventions), but we should be ready to commit it to the RooStats branch in ~1 day

For now, we’ve only made changes to the parser and the I/O parts of Histfactory, and have not altered the model-building itself

We have checked that our changes reproduce exactly the results from 5.32

We will prepare some simple examples for creating a model directly from c++/python