Improvements to the CAF

Carsten Burgard

ALU Freiburg

March 27th, 2013

New Makefiles

- refactored `HWWAnalysisCode` and `HWWlvlv_2012` Makefiles
  - compilation should run much faster now
  - benefit especially large for parallel compilation
  - use "`make -j` *n*" to enjoy super-fast `CAF` compilation, where *n* is the number of cores on your machine
- introduced new Makefile targets
  - type "`make doc`" to render `ROOT`-style html documentation into `doc` directory
- introduced `TQPATH` environment variable
  - put the following line into your .bashrc
    ```
    export TQPATH=/path/to/HWWAnalysisCode/on/your/machine
    ```
    required for some advanced macros/features, see following slide

tqroot

If you want to investigate an Analysis `ROOT`-file procued by the `CAF`

```
# root
# .L /some/obscure/path/libQFramework.so
# TFile* f = new TFile("/path/to/your/file/myfile.root")
```

- typing long pathnames is tedious and error-prone

```
# tqroot /path/to/your/file/myfile.root
```

- `tqroot` is a simple bash script that resides in `macros`
- it fires up root and autoloads the `libQFramework.so` before opening all files that are provided as arguments subsequently
- caveat: only works when called from the `HWWAnalysisCode` trunk directory

tqroot

If you want to investigate an Analysis `ROOT`-file procued by the `CAF`

```
# root
# .L /some/obscure/path/libQFramework.so
# TFile* f = new TFile("/path/to/your/file/myfile.root")
```

- typing long pathnames is tedious and error-prone

```
# tqroot /path/to/your/file/myfile.root
```

- `tqroot` is a simple bash script that resides in `macros`
- it fires up root and autoloads the `libQFramework.so` before opening all files that are provided as arguments subsequently
- caveat: only works when called from the `HWWAnalysisCode` trunk directory

. . . unless `TQPATH` is set, then it works from everywhere!

The "Code Conflict"

- all users want stable and performant code
- some users request/want new features
- some users depend on time-constant interfaces and backward-compatibility
- developers want to maintain as little code as possible

How can this be solved?

The “Code Conflict”

- all users want stable and performant code
- some users request/want new features
- some users depend on time-constant interfaces and backward-compatibility
- developers want to maintain as little code as possible

How can this be solved?

$\Rightarrow$ **modular** code

Modular code

Listing 1: modular code example

```
#ifndef MYFANCYNEWFEATURE
// standard-compliant, backward-compatible code goes here
#else
// new experimental variant goes here
#endif
```

- depending on the flag `MYFANCYNEWFEATURE`, the old/new code fragments are used
- flag can be set at compile time, allows for the same file to be compiled several times with different flags set to obtain "standard" and "experimental" binaries
- currently, two variants of `runHWWAna` are built by default:
  - `runHWWAna` is fully backwards-compatible
  - `runHWWAna-new` implements new/advanced features

Modular code: How to implement an “experimental” feature

If you implement an new feature and are afraid it might break backward-compatibility of the code

- think of a reasonable name for it
- wrap it in #ifdef YOURFEATURENAME/#endif
- compile with `-DYOURFEATURENAME` (“-D” for *define*) **and/or**
- insert `-DYOURFEATURENAME` into the Makefile

The appropriate location in the Makefile is the following block:

Listing 2: Makefile

```
$(RUN_HWWANA_NEW): $(OBJECTS) ↙
    ↪ $(SRC_DIR)/Run_HWW_Analysis_2012.cxx
  $(CXX) $(CXXFLAGS) -o $@ -g $^ $(LIBS) ↙
      ↪ -DYOURFEATURENAME
  @echo "===> your compilation of "$@" succeeded!"
```

From Run_HWW_Ana_2012.cxx...

Listing 3: Run_HWW_Ana_2012.cxx

```
if (doVBFStyle){
  printer->addProcess("sig/em/mh125/vbf + sig/em/mh125/WH + sig/em/mh125/ZH + sig/me/mh125/vbf + sig/me/mh125/WH + sig/me/mh125/ZH  ", "vbf+vh [125 GeV]");
  printer->addProcess("sig/em/mh125/ggf + sig/me/mh125/ggf", "ggf [125 GeV]");
}else{
  printer->addProcess("sig/em/mh125 + sig/me/mh125", "Signal [125 GeV]");
}
printer->addProcess("bkg/em/diboson/WW + bkg/me/diboson/WW", "$WW$");
printer->addProcess("bkg/em/diboson/NonWW + bkg/me/diboson/NonWW", "$WZ/ZZ/W\\gamma$");
```

. . .

Listing 4: Run_HWW_Ana_2012.cxx

```
TString sfLine_presel       = getScaleFactorLine(samples, ch, doVBFStyle, splitnonWW, "CutZVeto");
TString sfLine_METRel       = getScaleFactorLine(samples, ch, doVBFStyle, splitnonWW, "CutMETRel");
TString sfLine_SR_0jet = getScaleFactorLine(samples, ch, doVBFStyle, splitnonWW, "Cut_0jet");
TString sfLine_SR_1jet = getScaleFactorLine(samples, ch, doVBFStyle, splitnonWW, "Cut_1jet");
```

. . .

Listing 5: Run_HWW_Ana_2012.cxx

```
printer->addCut("||");
printer->addCut("CutWeights",  "blinding");
printer->addCut("CutLeptonPt", "lepton $p_{\\mathrm{T}}$");
printer->addCut("CutOSLeptons", "OS leptons");
```

Solution

- About 400 lines of code dedicated to configuring a given instance of `TQHWWCutflowPrinter2`
- should be done with a config file instead
- possible future improvement: similar configuration variant for the `TQHWWPlotter2`

Listing 6: Run_HWW_Ana_2012.cxx

```
TString chkey = ch;
if(ch.Contains("+"))
  chkey = "["+ch+"]";
printer->readProcessesFromFile(processFile,chkey);
printer->readCutsFromFile(cutFile);
```

Process config file syntax

```
||;
sig/%ch%/mh125;  Signal [125 GeV];
bkg/%ch%/diboson/WW;  $WW$;
bkg/%ch%/diboson/NonWW; $WZ/ZZ/W\gamma$;
```

Figure: `definitions/HWW_Cutflow_Processes.txt`

- cutflow columns separated by newlines
- semicolon separate arguments: *path to process*; *process title*;

TQCutflowPrinter2::addProcess("a","b") ⇒ **a**;**b**;

- no need for quotation
- don't escape backslashes of LaTeX control sequences

Cut config file syntax

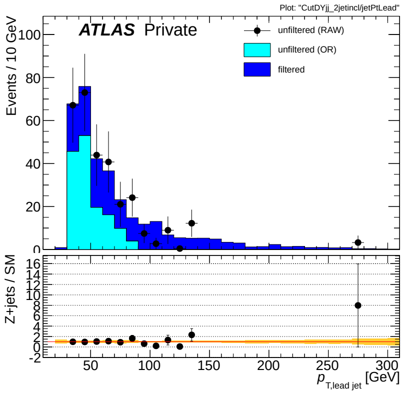
```
||;
CutWeights;  blinding;2
CutLeptonPt; lepton $p_{\mathrm{T}}$;
CutMll; $m_{\ell\ell} > 12,10$ GeV;1
CutZVeto; $Z$ veto (for $ee,\mu\mu$);0
```

Figure: `definitions/HWW_Cutflow_Cuts.txt`

- overall similar syntax
- additional argument: number n designates NF-policy

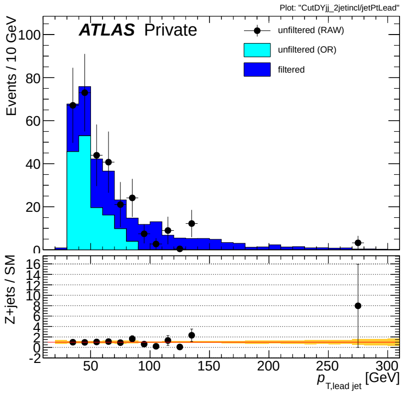
$n = 0$	do not ever print NFs
$n = 1$	print NFs whenever different from unity *(default)*
$n = 2$	print all NFs

Automagic plot scale adjustment

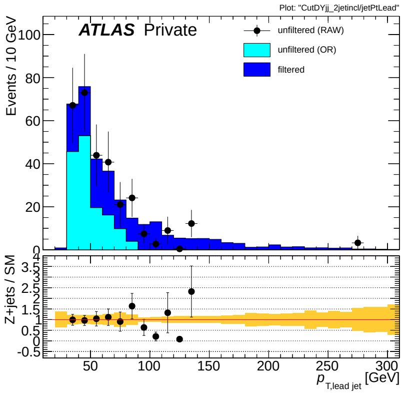


default behavior

Automagic plot scale adjustment



default behavior



```
TQHWWPlotter2::setTag("style.ratioMaxQerr",2);
```

How it works

- employs the new TQHWWPlotter2::getRange function
  - computes x and y-ranges of arbitrary `TGraphErrors`
  - takes additional argument `maxQerr`
  - loops over graph points, expands range for every point that is at most by a factor of `maxQerr` outside the current range
- quite technical, what do you need to know?
  - **large** `maxQerr`: more likely to **accept "outliers"**
  - **small** `maxQerr`: **aggressive range optimisation**
  - default corresponds to `maxQerr`$=\infty$, i. e. accepts all points
  - experimental evidence shows that reasonable values are typically within $1 \lesssim$ `maxQerr` $\lesssim 10$

Quick revision

Listing 7: Run_HWW_Ana_2012.cxx (again)

```
TString chkey = ch;
if(ch.Contains("+"))
  chkey = "["+ch+"]";
printer->readProcessesFromFile(processFile,chkey);
printer->readCutsFromFile(cutFile);
```

```
||;
sig/%ch%/mh125;  Signal [125 GeV];
bkg/%ch%/diboson/WW;  $WW$;
bkg/%ch%/diboson/NonWW; $WZ/ZZ/W\gamma$;
```

Figure: `definitions/HWW_Cutflow_Processes.txt` (again)

Arithmetic path expansion

```
                                    /sample/folder/path/a +
/sample/folder/path/[a+b-c] ⇒ /sample/folder/path/b -
                                    /sample/folder/path/c
```

- you can now use expressions like

  ```
  /bkg/[ee+mm]/Zjets/Z/Nom/[em+me]/*
  ```

  to read in your samples from the Sample Folder hierarchy!

Ideas

- cleanup `Run_HWW_Ana_2012.cxx` (!)
- config files for `TQHWWPlotter2`
- grand unification of config file syntax
  - we have a very configurable package
  - lots of config files are read in at runtime
  - all of them have a different syntax...
- class compatibility upgrade
  - currently: several active "class versions", e. g. `TQHWWPlotter2`
  - neccessary for backward compatibility to "old" `ROOT`-files
  - possible fix: class conversion features for streamer
- messaging services
  - printouts are useful, but clutter the output
  - implement messaging service, redirecting to various log files
- **input from your side is highly welcome**