MPF Documentation

hans

Jun 28, 2018

Contents:

MPF is a plotting framework that is meant to simplify the creation of typical "ATLAS-style" plots using pyROOT. Setup instructions and the code itself can be obtained here.

CHAPTER 1

Plotting from histograms

Plots can currently be created either from ROOT histograms. For example:

```
p = Plot()
p.registerHist(hist1, style="background", process="Bkg1")
p.registerHist(hist2, style="background", process="Bkg2")
p.registerHist(hist3, style="signal", process="Signal")
p.registerHist(hist4, style="data", process="Data")
p.SaveAs(outputFilename)
```

See `plotStore()` for details and examples.

CHAPTER 2

Plotting from ntuples

Plots can also be generated from flat ntuples (ROOT TTree). There is an extensive interface wrapping around TTree::Draw to ease the creation of multiple similiar histograms from different ntuples. For example:

```
tp = TreePlotter()
tp.addProcessTree("Bkg1", rootfile1, treename1, style="background")
tp.addProcessTree("Bkg2", rootfile2, treename2, style="background")
tp.addProcessTree("Signal", rootfile3, treename3, style="signal")
tp.addProcessTree("Data", rootfile4, treename4, style="data")
tp.plot(outputFilename, varexp=var, xmin=xmin, xmax=xmax, nbins=nbins)
```

See `treePlotter()` for details and examples.

Also, there is an option to create multiple histograms from a single TTree by looping the tree only once. For example:

```
tp.registerPlot(outputFilename1, varexp=var1, xmin=xmin1, xmax=xmax1, nbins=nbins1)
tp.registerPlot(outputFilename2, varexp=var2, xmin=xmin2, xmax=xmax2, nbins=nbins2)
tp.plotAll()
```

CHAPTER 3

Global options

Most options are directly passed to the Functions/Classes that create the plots. A few, more global options can also be set via the `MPF.globalStyle` module.

CHAPTER 4

Memoization

All histograms that are created from ntuples are cached across multiple executions (using the meme module), so once created, modifications to the plot (e.g. style) can be made without having to loop the ntuples again. This is very useful, however currently meme is not thread-safe - so in case you are about to run MPF in parrallel in the same folder you should deactivate the meme module:

```
from MPF import meme
meme.setOptions(deactivated=True)
```

By default the cache will be stored in a folder `_cache` in the directory your code is executed. This can be changed by:

```
meme.setOptions(overrideCache=myCachePath)
```

To enforce re-execution of cached functions either delete the cache folder or use:

```
meme.setOptions(refreshCache=True)
```

CHAPTER 5

Verbosity

MPF uses the builtin logging from python. Both MPF and meme configure the logger already at module level (which they probably shouldn't and might change in the future). You can set the logging levels by retrieving the loggers. For example to deactivate the INFO messages:

```
import logging
logging.getLogger("MPF").setLevel(logging.WARNING)
logging.getLogger("meme").setLevel(logging.WARNING)
```

The submodules of MPF use child loggers of the `MPF` logger. The naming scheme is `MPF.MPF.<submodule-name>`. For example to activate the DEBUG messages only for the `histProjector` module do:

```
logging.getLogger("MPF.MPF.histProjector").setLevel(logging.DEBUG)
```

CHAPTER 6

Table of contents

6.1 MPF package

6.1.1 Submodules

CHAPTER 7

Indices and tables

- genindex
- modindex
- search