

Notes on the **FSRoot** Package

Ryan Mitchell

February 24, 2019

Abstract

FSRoot is a set of utilities to help manipulate information about different Final States (FS) produced in particle physics experiments. The utilities are built around the CERN **ROOT** framework. This document provides an introduction to **FSRoot**.

Contents

1	Installation and Initial Setup	1
2	Basic Operations: the FSBasic directory	2
2.1	Basic Conventions	2
2.2	Basic Histogram Utilities: the FSHistogram class	3
2.3	Basic Tree Utilities the FSTree class	4
3	Final State Operations: the FSMode directory	4
3.1	Mode Numbering and Conventions	4
3.2	Mode Information: the FSModeInfo class	6
3.3	Collections of Modes: the FSModeCollection class	6
3.4	Histograms for Multiple Modes: the FSModeHistogram class	6
3.5	Operations on Multiple Trees: the FSModeTree class	6
4	Fitting Utilities: the FSFit directory	6

1 Installation and Initial Setup

Instructions for installation and initial setup:

(1) Download the source:

```
> git clone https://github.com/remitch66/FSRoot.git FSRoot
```

(2) Set the location of **FSRoot** in your login shell script (e.g. `.cshrc`):

```
setenv FSROOT [xxxxx]/FSRoot
```

(3) Also probably add the `FSRoot` directory to `$DYLD_LIBRARY_PATH` and `$LD_LIBRARY_PATH`. This allows you to compile code including `FSRoot` functions. For example:

```
setenv DYLD_LIBRARY_PATH $DYLD_LIBRARY_PATH:$FSROOT
setenv LD_LIBRARY_PATH $LD_LIBRARY_PATH:$FSROOT
```

(4) There is usually a `.rootrc` file in your home directory that `ROOT` uses for initialization. Add lines like these to `.rootrc`, which tell `ROOT` the location of `FSRoot`:

```
Unix.*.Root.DynamicPath: .:$(FSROOT):$(ROOTSYS)/lib:
Unix.*.Root.MacroPath: .:$(FSROOT):
```

(5) Now when you open `ROOT`, the `FSRoot` utilities should be loaded and compiled – you should see a message saying “Loading the `FSRoot` Macros” along with the output of the compilation. The loading and compiling is done in the file `rootlogon.C`. This file also sets up default styles, which are not essential. Since these might conflict with styles you have defined elsewhere, it could be worthwhile to tweak or remove these.

2 Basic Operations: the `FSBasic` directory

2.1 Basic Conventions

Some `FSRoot` operations on variables within a `ROOT TTree` assume a particular format within the tree. Four-vectors are assumed to take the form:

```
[AB]EnP[CD], [AB]PxP[CD], [AB]PyP[CD], [AB]PzP[CD]
```

where `[CD]` is a particle label (often “1”, “2”, “3”, “2a”, etc., but also “CM” or “B” or other) and `[AB]` labels the type of four-vector (for example, “R” for raw or “K” for kinematically fit or “MC” for Monte Carlo, etc.). The `FSRoot` code does not assume anything about the conventions for `[AB]`. The final state utilities described in Sec. 3 require the particles appear in a given order and assume a numbering convention for `[CD]`, described in Sec. 3.1. The utilities described in this section do not assume a specific form for `[CD]`.

Variable names for run numbers, event numbers, and the χ^2/dof from a kinematic fit are also hardcoded in a couple of places (like in the functions that rank combinations within an event by χ^2/dof):

```
Run, Event, Chi2DOF
```

Maybe a future version of `FSRoot` could make these formats customizable.

2.2 Basic Histogram Utilities: the FSHistogram class

Basic histogram functions are provided by the FSHistogram class in the FSBasic directory. Like many other functions within FSRoot, the functions within the FSHistogram class are static member functions, so there is never a need to deal with instances of FSHistogram.

The basic functionality is through the FSHistogram::getTH1F and FSHistogram::getTH2F classes. Here are example uses that can be run from either the ROOT command line or from a macro (see Examples/Intro/intro.C). The first draws a 1d histogram and the second draws a 2d histogram. The third argument is the variable to plot; the fourth holds the number of bins and bounds.

```
> FSHistogram::getTH1F(FileName,TreeName,"Chi2DOF","(60,0.0,6.0)","")->Draw();
> FSHistogram::getTH2F(FileName,TreeName,"Chi2DOF:Event",
    "(100,0.0,1.0e6,60,0.0,6.0)","")->Draw("colz");
```

Cuts can be added in the fifth argument:

```
> FSHistogram::getTH1F(FileName,TreeName,"Chi2DOF",
    "(60,0.0,6.0)","Chi2DOF<5.0")->Draw();
```

The variable and cut arguments can contain shortcuts. For example, MASS(I,J) is expanded into the total invariant mass of particles I and J, where I and J are not necessarily numbers (they are the [CD] in Sec. 2.1). Characters in front of MASS (for example) are prepended to the variable names (the [AB] in Sec. 2.1). This is all done in the function FSTree::expandVariable, which can also be used on its own to explicitly see what it is doing. See FSTree::expandVariable for more options beyond MASS (for example, RECOILMASS, MOMENTUM, COSINE, etc.). Examples:

```
> FSHistogram::getTH1F(FileName,TreeName,"MASS(1,2)",
    "(60,0.0,6.0)","Chi2DOF<5.0&&MASS(1,2)>1.0")->Draw();
> cout << FSTree::expandVariable("MASS(1,2)+MASS(2,3)") << endl;
> cout << FSTree::expandVariable("ABMASS(1,C)") << endl;
> cout << FSTree::expandVariable("RECOILMASS(CM;D,2)") << endl;
```

Histograms are automatically cached so they are made only once. To save histograms at the end of a session, use the function:

```
> FSHistogram::dumpHistogramCache();
```

To read in the cache at the beginning of a session:

```
> FSHistogram::readHistogramCache();
```

To clear a cache from memory during a session:

```
> FSHistogram::clearHistogramCache();
```

To see more verbose output during a session:

```
> FSControl::DEBUG = true;
```

2.3 Basic Tree Utilities the FSTree class

The **FSTree** class is also located in the **FSBasic** directory and provides basic utilities to operate on trees. Besides the static **FSTree::expandVariable** member function mentioned in Sec. 2.2, the most useful function is for skimming trees. For example:

```
> FSTree::skimTree(inputFileName, inputTreeName, outputFileName,
    "Chi2DOF<5.0");
```

will take the tree named **inputTreeName** from the file **inputFileName**, loop over all events and select only those that pass the cut on **Chi2DOF**, then output the selected events to the file named **outputFileName** in a tree with the same name as the input tree. The shortcuts mentioned in Sec 2.2 can also be used here, for example:

```
> FSTree::skimTree(inputFileName, inputTreeName, outputFileName,
    "abs(MASS(1,2)-0.5)<0.1");
```

3 Final State Operations: the FSMode directory

3.1 Mode Numbering and Conventions

A “final state” (also called “mode”) is made from a combination of: $\Lambda(\rightarrow p\pi^-)$, $\bar{\Lambda}(\rightarrow \bar{p}\pi^+)$, e^+ , e^- , μ^+ , μ^- , p , \bar{p} , $\eta(\rightarrow \gamma\gamma)$, γ , K^+ , K^- , $K_S^0(\rightarrow \pi^+\pi^-)$, π^+ , π^- , $\pi^0(\rightarrow \gamma\gamma)$.

As strings, these final state particles are given as: $\Lambda(\rightarrow p\pi^-) \equiv \text{Lambda}$, $\bar{\Lambda}(\rightarrow \bar{p}\pi^+) \equiv \text{ALambda}$, $e^+ \equiv \text{e+}$, $e^- \equiv \text{e-}$, $\mu^+ \equiv \text{mu+}$, $\mu^- \equiv \text{mu-}$, $p \equiv \text{p+}$, $\bar{p} \equiv \text{p-}$, $\eta(\rightarrow \gamma\gamma) \equiv \text{eta}$, $\gamma \equiv \text{gamma}$, $K^+ \equiv \text{K+}$, $K^- \equiv \text{K-}$, $K_S^0(\rightarrow \pi^+\pi^-) \equiv \text{Ks}$, $\pi^+ \equiv \text{pi+}$, $\pi^- \equiv \text{pi-}$, $\pi^0(\rightarrow \gamma\gamma) \equiv \text{pi0}$.

In a **TTree**, the final state particles should be listed in the order they are given above. The numbering for the [CD] (Sec. 2.1) starts at “1”. For final state particles that decay ($\Lambda(\rightarrow p\pi^-) \equiv \text{Lambda}$, $\bar{\Lambda}(\rightarrow \bar{p}\pi^+) \equiv \text{ALambda}$, $\eta(\rightarrow \gamma\gamma) \equiv \text{eta}$, $K_S^0(\rightarrow \pi^+\pi^-) \equiv \text{Ks}$, $\pi^0(\rightarrow \gamma\gamma) \equiv \text{pi0}$), the four-momenta of the decay particles are listed using “a” and “b” in the same order as above or ordered by energy for identical particles.

For example, for the final state $\gamma K^+ K_S^0 \pi^+ \pi^- \pi^0$, the four-momenta are:

```
EnP1 PxP1 PyP1 PzP1      (for the gamma)
```

EnP2	PxP2	PyP2	PzP2	(for the K+)
EnP3	PxP3	PyP3	PzP3	(for the Ks)
EnP3a	PxP3a	PyP3a	PzP3a	(for the pi+ from Ks)
EnP3b	PxP3b	PyP3b	PzP3b	(for the pi- from Ks)
EnP4	PxP4	PyP4	PzP4	(for the pi+)
EnP5	PxP5	PyP5	PzP5	(for the pi-)
EnP6	PxP6	PyP6	PzP6	(for the pi0)
EnP6a	PxP6a	PyP6a	PzP6a	(for the higher energy gamma from pi0)
EnP6b	PxP6b	PyP6b	PzP6b	(for the lower energy gamma from pi0)

Every final state can be specified in three different ways:

(1) `pair<int,int> modeCode`: a pair of two integers (`modeCode1`, `modeCode2`) that count the number of particles in a decay mode.

```

modeCode1 = abcdefg
a = # gamma
b = # K+
c = # K-
d = # Ks
e = # pi+
f = # pi-
g = # pi0
modeCode2 = abcdefghi
a = # Lambda
b = # ALambda
c = # e+
d = # e-
e = # mu+
f = # mu-
g = # p+
h = # p-
i = # eta

```

(2) `TString modeString`: a string version of `modeCode1` and `modeCode2` in the format “`modeCode2_modeCode1`”. It can contain a prefix (for example, “FS” or “EXC” or “INC” or anything longer) that isn’t used here, but can help with organization elsewhere.

(3) `TString modeDescription`: a string with a list of space-separated particle names (for example “K+ K- pi+ pi+ pi- pi-”). The final state particles can appear in any order.

For example, the final state $\gamma K^+ K_S^0 \pi^+ \pi^- \pi^0$ has `modeCode1 = 1101111`, `modeCode2 = 0`, `modeString = "ANYTHING_0_1101111"`, `modeDescription = "gamma K+ Ks pi+ pi- pi0"`.

3.2 Mode Information: the FSModeInfo class

```
> FSModeInfo mi("K+ K- K+ K- pi+ pi- pi0 eta");
> mi.modeString(); // ==> "1_220111"
> mi.modeCode1(); // ==> 220111
> mi.modeCode2(); // ==> 1
> mi.modeString("this is the code: (MODECODE1,MODECODE2)");
// ==> "this is the code: (220111,1)"

> mi.modeCombinatorics("ABC[K+] DEF[K-]",true);
// ==> *** MODE COMBINATORICS TEST ***
// ==> ABC2 DEF4
// ==> ABC3 DEF4
// ==> ABC2 DEF5
// ==> ABC3 DEF5
// ==> *****

> mi.modeCuts("OR((ABC[K+]+DEF[K-])>0)");
// ==> "((ABC2+DEF4)>0)||((ABC3+DEF4)>0)||((ABC2+DEF5)>0)||((ABC3+DEF5)>0))"
```

3.3 Collections of Modes: the FSModeCollection class

3.4 Histograms for Multiple Modes: the FSModeHistogram class

3.5 Operations on Multiple Trees: the FSModeTree class

4 Fitting Utilities: the FSFit directory

The fitting utilities (contained in the directory FSFit) work, but are still under development. See the examples in the directory `Examples/Fitting` to get the general idea.