The RooRarFit Document

The RooRarFit Document

Lei Zhang

Installation

To install RooRarFit, download the tar file from SF.net (`http://sourceforge.net/projects/rarfit`), unpack and compile. Make sure $ROOTSYS is defined first and that the ROOT libraries are in your library path.

```
> echo $ROOTSYS; echo $LD_LIBRARY_PATH
> tar -zxvf RooRarFit.VXX.YY.ZZZ.tar.gz
> cd RooRarFit
> make bin
```

This will create an executable in ./tmp.

```
> ./tmp/rarFit
```

Preface

What is RooRarFit?

RooRarFit is a general Maximum Likelihood (ML) fitting package based on ROOT (`http://root.cern.ch/`) / RooFit (`http://root.cern.ch/drupal/content/roofit`). It is an application package making use of ROOT/RooFit so physicists can be virtually free from direct ROOT/RooFit coding.

Why not use RooFit directly?

A ill-designed tool sometimes can really limit analysis capacity, so for best flexibility and availability, we could be better off using ROOT/RooFit directly. However this fitter requires no direct C++ coding, which is highly preferable for the purpose of productivity.

By using plain text configuration files, it is virtually free of coding for final users, so people do not need much programming experience to use it. Debugging an analysis is quite frustrating and if people have to compile code for individual analyses, he has to make sure the code for every mode is correct and has to debug it if something is wrong.

General Idea of RooRarFit

The idea of the fitter is that it acts as a wrapper of RooFit so that the fitter is driven by a readable text configuration files. It should be easy to implement, easy to improve, and easy to expand. If the goal as a wrapper is clear, all of the features are easy to realize.

The configuration file has three main parts, each having one or more sections.

1. Dataset Definition
   This defines the dataset and reads in all the datasets.
2. PDF Configuration
   All the PDFs are defined here. It can have any number of components in the ML function with any number of variables. Each PDF has its own configuration section. Fitter classes are wrappers of those in RooFit, and PDFs like Gaussian, TwoGauss, BifurGauss, Polynomial, etc..., are implemented, as well as combination operations like Add, Prod, etc. So it is very easy to build new PDF, through a configuration file only!
3. Fitter Action Section
   After every PDF is created, this part of the configuration file is used to direct the fitter to finish its job. Possible actions include fitting the PDFs (pdfFit), fitting the model (mlFit), performing Monte Carlo Toy studies (toyStudy), projection plotting (ProjAct), scanning the log-likelihood distribution (ScanAct), making sPlots (SPlotAct) and combing statitsical and systematic errors (CombineAct). For each fitting job, one can decide which action, or what action combinations, to perform.

1 RooRarFit Implementation

The source codes of the new mlFitter can be classified into three groups:

1. Auxiliary Classes, Files, Main Program and Scripts

   - rarFit.cc (`http://rarfit.sourceforge.net/html/rarFit_8cc.html`), the main programme
     It first creates a rarDatasets (`#item_rarDatasets`) object through which all the datasets are read in, then it sets the master (`mlFitter`) section name and instantiates `mlFitter` class, rarMLFitter (`#item_rarMLFitter`), so that all the PDF are created, and finally it sets toyID (random seed) and calls run() of the fitter to finish the job. If successfully compiled and linked, type `rarFit` to see short help page like:

     ```
     rarFit [-options] <RooRarFit_Config_file>
             -h this help page
             -D <data input section> (default "Dataset Input")
             -C <mlFitter config section> (default "mlFitter Config")
             -A <fitter action section> (default "Fitter Action")
             -t <toy job id> (default 0)
             -n <toyNexp> (default 0, use config)
             -d <toy dir> (default .toyData)
     ```

     You can set environment variables to control the behavior of the fitter:

     - `PARAMDIR`: The dir for all param files (default `.params`)
     - `RESULTDIR`: The dir for all root files (default `results`)
     - `RANDOMSEEDBASE`: The random seed will be based on this value and toy job ID. It should be an integer (default 0).

     Environment variables are intended for scripting, so it is NOT recommended to use them at command line interactively.

   - `submitToy`, script to submit batch (toy) jobs
     This is a Perl script `submitToy` used to submit batch jobs on many BaBar sites, *slac, ral, cu-boulder, ed.* It is called `submitToy`, but it can submit any kind of `rarFit` jobs (actions) which can be split and the results of which can be merged. For example, toy study, scanPlot (multi-dimensional), and projPlot (for LLR plot), can be split and submitted using this script. Type `submitToy -h` for help page:

     ```
     Usage : submitToy [options] configFile

       where configFile is a RooRarFit config file and
       options are (defaults in parenthesis):

          -h           : this help message
          -n nExp      : total number of experiments
          -j nJobs     : number of jobs to run
          -D dsi       : toyAct dsi from config
          -C pdfConfSec: toyAct master PDF section from config
          -A action    : toyAct name from config
     ```

```
    -q queue     : specific queue to submit to
    -d jobDir    : directory for the jobs (toyJob)
    -t           : testing, not submit the jobs in the queue

 This script will take a config file that configures
a RooRarFit job and split it into a number of separate jobs.
The jobs are then submitted to the queue of your choice.

Example:  submitToy -n 500 -j 10 -A eToyAct -d etoy_omks omks.config
submits 10 jobs based on the eToyAct action in the
omks.config file
```

- rarStrParser (`http://rarfit.sourceforge.net/html/classrarStrParser.html`)
  It breaks a string into tokens separated by spaces. Characters inside quote(") are considered one token.
- rarVersion.hh (`http://rarfit.sourceforge.net/html/rarVersion_8hh.html`)
  In this header file, several C++ macros to deal with version related codes are defined.
- rarMinuit (`http://rarfit.sourceforge.net/html/classrarMinuit.html`)
  It is adopted for contour plot.
- rarNLL (`http://rarfit.sourceforge.net/html/classrarNLL.html`)
  It deals with Negative Log-Likelihood (NLL) NLL curves to get significance, upper limits, etc. By using analytical integral of parabolic fit wrt 3 points in the NLL curve, the accuracy is much more improved than using just linear fit.

2. Dataset Classes

   - rarDatasetDef (`http://rarfit.sourceforge.net/html/classrarDatasetDef.html`)
     It defines the format of dataset, ie, how many fields, the type of each field, etc.
   - rarDatasets (`http://rarfit.sourceforge.net/html/classrarDatasets.html`)
     It first instantiates a rarDatasetDef (`#item_rarDatasetDef`) object to get the dataset definitions, then it reads in and holds all the datasets from ASCII or root files. It also holds datasets derived from those primary datasets.

3. PDF Classes

   - rarBasePdf (`http://rarfit.sourceforge.net/html/classrarBasePdf.html`), base class of PDF builder.
   - rarCompBase (`http://rarfit.sourceforge.net/html/classrarCompBase.html`), base class for composite PDF builder.
   - rarProd (`http://rarfit.sourceforge.net/html/classrarProd.html`), product PDF builder, (RooProdPdf (`http://root.cern.ch/root/html/RooProdPdf.html`))
   - rarAdd (`http://rarfit.sourceforge.net/html/classrarAdd.html`), add PDF/model builder, (RooAddPdf

(http://root.cern.ch/root/html/RooAddPdf.html) / RooAddModel (http://root.cern.ch/root/html/RooAddModel.html))

- rarMLPdf (http://rarfit.sourceforge.net/html/classrarMLPdf.html), extended AddPdf as prototype PDF to build final mlFitter, sub-class of rarAdd (#item_rarAdd)
- rarMLFitter (http://rarfit.sourceforge.net/html/classrarMLFitter.html), mlFitter class, sub-class of rarCompBase (#item_rarCompBase), which is instantiated by the main program in rarFit.cc (#item_rarFit.cc), then builds PDFs from top to bottom, and creates simultaneous fit model through RooSimPdfBuilder (http://root.cern.ch/root/html/RooSimPdfBuilder.html) if required, and finally finishes its job through run() function called by the main program.
- Wrappers of other RooFit PDF classes
  - rarSimPdf (http://rarfit.sourceforge.net/html/classrarSimPdf.html), (RooSimultaneous (http://root.cern.ch/root/html/RooSimultaneous.html))
  - rarBallack (http://rarfit.sourceforge.net/html/classrarBallack.html)
  - rarBinned (http://rarfit.sourceforge.net/html/classrarBinned.html)
  - rarCruijff (http://rarfit.sourceforge.net/html/classrarCruijff.html)
  - rarFlatte (http://rarfit.sourceforge.net/html/classrarFlatte.html)
  - rarVoigtian (http://rarfit.sourceforge.net/html/classrarVoigtian.html)
  - rarRelBreitWigner (http://rarfit.sourceforge.net/html/classrarRelBreitWigner.html)
  - rarGounarisSakurai (http://rarfit.sourceforge.net/html/classrarGounarisSakurai.html)
  - rarExp (http://rarfit.sourceforge.net/html/classrarExp.html), (Exponential (http://root.cern.ch/root/html/RooExponential.html))
  - rarGaussian (http://rarfit.sourceforge.net/html/classrarGaussian.html), (Gaussian (http://root.cern.ch/root/html/RooGaussian.html) / BreitWigner (http://root.cern.ch/root/html/RooBreitWigner.html))
  - rarTwoGauss (http://rarfit.sourceforge.net/html/classrarTwoGauss.html), (TwoGaussian)
  - rarTriGauss (http://rarfit.sourceforge.net/html/classrarTriGauss.html), (TripleGauss / TripleGaussModel / GexpShape)

- rarBifurGauss (http://rarfit.sourceforge.net/html/classrarBifurGauss.html), (BifurGauss (http://root.cern.ch/root/html/RooBifurGauss.html))
- rarCBShape (http://rarfit.sourceforge.net/html/classrarCBShape.html), (Crystal Ball Shape (http://root.cern.ch/root/html/RooCBShape.html))
- rarPoly (http://rarfit.sourceforge.net/html/classrarPoly.html), (Polynomial (http://root.cern.ch/root/html/RooPolynomial.html) / RooChebychev (http://root.cern.ch/root/html/RooChebychev.html))
- rarArgusBG (http://rarfit.sourceforge.net/html/classrarArgusBG.html), (ArgusBG (http://root.cern.ch/root/html/RooArgusBG.html))
- rarStep (http://rarfit.sourceforge.net/html/classrarStep.html), (ParametricStepFunction (http://root.cern.ch/root/html/RooParametricStepFunction.html))
- rarKeys (http://rarfit.sourceforge.net/html/classrarKeys.html), (Keys (http://root.cern.ch/root/html/RooKeysPdf.html) / 2DKeys (http://root.cern.ch/root/html/Roo2DKeysPdf.html))
- rarGeneric (http://rarfit.sourceforge.net/html/classrarGeneric.html), (Generic (http://root.cern.ch/root/html/RooGenericPdf.html))
- rarGaussModel (http://rarfit.sourceforge.net/html/classrarGaussModel.html), (GaussModel (http://root.cern.ch/root/html/RooGaussModel.html))
- rarDecay (http://rarfit.sourceforge.net/html/classrarDecay.html), (BCPGenDecay (http://root.cern.ch/root/html/RooBCPGenDecay.html) / BDecay (http://root.cern.ch/root/html/RooBDecay.html) / RooDecay (http://root.cern.ch/root/html/RooDecay.html))
- rarHistPdf (http://rarfit.sourceforge.net/html/classrarHistPdf.html), (RooHistPdf (http://root.cern.ch/root/html/RooHistPdf.html))
- rarUniform (http://rarfit.sourceforge.net/html/classrarUniform.html), (RooUniform (http://root.cern.ch/root/html/RooUniform.html))

- rarConfig (http://rarfit.sourceforge.net/html/classrarConfig.html), base class for dataset and PDF classes

  Dataset and PDF classes (the 2nd and 3rd items above) are both derived from base class, rarConfig (#item_rarConfig), which defines common data and functions to construct dataset/PDF objects from config files. The actual creator for objects of RooRealVar (http://root.cern.ch/root/html/RooRealVar.html), RooConstVar (http://root.cern.ch/root/html/RooConstVar.html), RooUnblindPrecision (http://root.cern.ch/root/html/RooUnblindPrecision.html), RooCategory

(http://root.cern.ch/root/html/RooCategory.html), RooMappedCategory (http://root.cern.ch/root/html/RooMappedCategory.html), RooThresholdCategory (http://root.cern.ch/root/html/RooThresholdCategory.html), RooStringVar (http://root.cern.ch/root/html/RooStringVar.html), RooFormulaVar (http://root.cern.ch/root/html/RooFormulaVar.html), is the createAbsVar (http://rarfit.sourceforge.net/html/classrarConfig.html#b12) function of rarConfig. Creation of RooDataSet (http://root.cern.ch/root/html/RooDataSet.html) is done by function createDataSet (http://rarfit.sourceforge.net/html/classrarConfig.html#b15), and all RooRarFit PDF objects except rarMLFitter (#item_rarMLFitter) are created by its createPdf (http://rarfit.sourceforge.net/html/classrarConfig.html#b16).

To add new type of RooRarFit PDF, one has to do two things to make new type of RooRarFit PDF available. First he needs to create a new class inherited from rarBasePdf (#item_rarBasePdf), second he adds an entry in function createPdf (http://rarfit.sourceforge.net/html/classrarConfig.html#b16) so the new class can be instantiated through the standard creation mechanism in RooRarFit.

To make it easier for user to add their own PDFs, empty RooRarFit PDF class `rarUsrPdf` can be modified to have quick access to PDFs not defined currently with RooRarFit. See Section 2.2.31 [rarUsrPdf Configuration], page 45.

2 RooRarFit Configuration

2.1 Dataset Configuration

The dataset configs have two parts: one is to define the structure of dataset entry, the other one is to read in and store the datasets.

2.1.1 Dataset Definition Section

The container of dataset column definitions, an object of class rarDatasetDef (`#item_rarDatasetDef`), is instantiated by datasets object, rarDatasets (`#item_rarDatasets`), and its configuration section is specified in dataset input section with configuration `dsdSec` See Section 2.1.2 [Dataset Input], page 15. If `dsdSec` is not specified, the default dataset definition section is `[Dataset Definition]`.

- `Fields = <field01> <field02> ... <fieldN>`
- `<field01> = <varType> ...`
- `<field02> = <varType> ...`
- `...`
- `<fieldN> = <varType> ...`
  `Fields` is a string containing all the names of the variables in data files. The order of the variables should be the same as in the ascii data file. `<varType>` can be `RooRealVar`, `RooCategory`, or `RooStringVar`.
- `AddOns = <addon01> ... <addonM>`
- `<addon01> = <derivedVarType> ...`
- `...`
- `<addonM> = <derivedVarType> ...`
  Optional `AddOns` specifies derived columns for a dataset after it is read in from data file. `<addon01>`, `...`, `<addonM>` must be derived from other variables declared in `Fields`. `<derivedVarType>` can be `RooFormulaVar`, `RooMappedCategory`, `RooThresholdCategory`, `RooSuperCategory`.

Because all these variables declared in this section are globally accessible, they should have unique and meaningful names. (With the naming schema (`#item_FullNameSchema`) for dataset definition section, the configuration name *is* its final, unique variable name if the optional full name is not given.)

A sample configuration section for dataset definition is shown below:

```
[Dataset Definition]
// Fields defined for datasets
Fields = onOffRes cosT de mes hiEvtID loEvtID

// Definition for each field
onOffRes= RooCategory "on/off Ups(4S) resonance" 3 Unknown 0 OnRes 1 OffRes 2
cosT    = RooRealVar "cosT" 0 1. B(100)
de      = RooRealVar "#Delta E" -0.01 0.08 B(45) "GeV"
mes     = RooRealVar "M_{es}" 5.20 5.29 B(45) "GeV"
```

```
kltype  = RooCategory "kltype" 3 emc 0 ifr 1 both 2
hiEvtID = RooStringVar "TS upperID"
loEvtID = RooStringVar "TS lowerID"
```

Since many configuration files (for the same analysis or similar analyses) may share the same dataset definition section, it is advisable to put the contents of this section into a separate configuration file and include that file into this section. For example,

```
[Dataset Definition]
// The actual configuration items (shown in the previous example) are in dsd.config
include dsd.config
```

This setting also makes the main configuration file more concise.

All these `RooAbsArg` variables are created using rarConfig::createAbsVar (`#item_rarConfig`).

- General rules to create variable
  A variable (can be any sub-class of `RooAbsArg`) can be used as observable (as in dataset), or parameter (as pdf parameter). In RooRarFit, all those variables are defined with the same syntax. It is also supposed that any created variable belongs to some RooRarFit object, which is instantiated from any sub-classes of rarConfig. So by default the name of variable is the concatenation of its configuration item name and the name of its owning RooRarFit object. For example, if the configuration is `myVar = <RooRealVar output>`, and the object creates this variable is `myPdf`, then the name of the created `RooRealVar` will be `myPdf_myVar`. This default behavior can be changed by explicitly giving a name as the first argument of the variable configuration item. For example, if the configuration is `myVar = myVarName ...`, the name of the `RooRealVar` will be `myVarName`. The next argument is the type of the variable, `RooRealVar` (optional), `RooConstVar`, `RooUnblindOffset`, `RooUnblindPrecision`, `RooCategory`, `RooMappedCategory`, `RooThresholdCategory`, `RooSuperCategory`, `RooStringVar`, or `RooFormulaVar`. The next argument is the title of the variable. If the definition has its own name as the first argument, the final title is the one in the configuration string, if the variable does not have name as the first argument, the final title will be the concatenation of title defined here and the title of its owning RooRarFit object. Please make sure the title is meaningful and concise, because it will be printed as label if the variable (as pdf parameter) is in a plot. So special character in ROOT for LaTex output will be preferred to get better output. The number and format of the remaining arguments depend on the type of the variable and how it will be created.

  To refer a variable to an existing variable, just give the existing variable's name as the only argument in the configuration entry:

  ```
  myVar = the_Existing_Var
  ```

  Unless `the_Existing_Var` is defined within the same section of `myVar`, `the_Existing_Var` should be a full name.

- Syntax to create `RooRealVar` (preferred)
  - `<varConfigName> = [[N] <finalName>] [[T] "<varTitle>"] [[U] "<unit>"] <RooRealVar Output Format>`
    You can specify optional `<finalName>`, `"<varTitle>"`, or `"<unit>"`, or you can have the `RooRealVar` output stream as the only part. This way of defining

RooRealVar is preferable for PDF parameters. If the order of name, title and unit is not the same as default, you need to precede them with N, T, or U to specify which is which.

- Syntax to create RooRealVar
  - <varConfigName> = [<finalName>] RooRealVar <varTitle> [C] <val> <min> <max> B(<nBin>) "<unit>"
  - <varConfigName> = [<finalName>] RooRealVar <varTitle> [C] <val> <min> <max> B(<nBin>)
  - <varConfigName> = [<finalName>] RooRealVar <varTitle> [C] <val> <min> <max> "<unit>"
  - <varConfigName> = [<finalName>] RooRealVar <varTitle> [C] <min> <max> B(<nBin>) "<unit>"
  - <varConfigName> = [<finalName>] RooRealVar <varTitle> [C] <min> <max> B(<nBin>)
  - <varConfigName> = [<finalName>] RooRealVar <varTitle> [C] <min> <max> "<unit>"
  - <varConfigName> = [<finalName>] RooRealVar <varTitle> [C] <val> <min> <max>
  - <varConfigName> = [<finalName>] RooRealVar <varTitle> [C] <min> <max>
  - <varConfigName> = [<finalName>] RooRealVar <varTitle> [C] <val> "<unit>"
  - <varConfigName> = [<finalName>] RooRealVar <varTitle> [C] <val>
    The meaning of all the fields are pretty straightforward. If the optional C is specified, the variable will be set to constant after it is created. B(<nBin>) defines the binning of this variable when it is plotted. Once <finalName> is given, the variable name will be set to it. Make sure that this <finalName> is unique globally, because createAbsVar will not create a variable twice with the same name, instead it returns a pointer to the already created variable.
- Syntax to create RooConstVar
  - <varConfigName> = [<finalName>] RooConstVar <varTitle> <val>
- Syntax to create RooUnblindOffset
  - <varConfigName> = [<finalName>] RooUnblindOffset "<varTitle>" "<blindString>" <scale> <blindValue>
  - <varConfigName> = [<finalName>] RooUnblindOffset "<varTitle>" "<blindString>" <scale> <blindValue> <blindState>

  <blindValue> is the RooAbsReal variable to blind, <blindState> is a RooAbsCategory to specify the blind state.
- Syntax to create RooUnblindPrecision
  - <varConfigName> = [<finalName>] RooUnblindPrecision "<varTitle>" "<blindString>" <centralValue> <scale> <blindValue> <sin2betaMode>

- `<varConfigName> = [<finalName>] RooUnblindPrecision "<varTitle>" "<blindString>" <centralValue> <scale> <blindValue> <blindState> <sin2betaMode>`

`<blindValue>` is the `RooAbsReal` variable to blind, `<blindState>` is a `RooAbsCategory` to specify the blind state, `<sin2betaMode>` is a boolean (`kTRUE` or `kFALSE`).

- Syntax to create `RooCategory`
  - `<catConfigName> = [<finalName>] RooCategory <catTitle> useIdx <Name1> <Idx1> <Name2> <Idx2> ...`
  - `<catConfigName> = [<finalName>] RooCategory <catTitle> noIdx <Name1> <Name2> ...`

  In the first form, `useIdx` is set, tokens following it are category type and index pairs. In the second form, `noIdx` is set, all tokens following it are category types and the indices are set to default values.
- Syntax to create `RooMappedCategory`
  - `<catConfigName> = [<finalName>] RooMappedCategory <catTitle> useIdx <inputCat> <defaultCatName> <defaultCatIdx> <inKeyRegExp1> <outKeyName1> <outKeyNum1> ...`
  - `<catConfigName> = [<finalName>] RooMappedCategory <catTitle> noIdx <inputCat> <defaultCatName> <inKeyRegExp1> <outKeyName1> ...`

  In the first form, `useIdx` is set, all the indices for mapped category types should be given explicitly. In the second form, `noIdx` is set, all the indices are set to default values.
- Syntax to create `RooThresholdCategory`
  - `<catConfigName> = [<finalName>] RooThresholdCategory <catTitle> useIdx <inputVar> <defaultCatName> <defaultCatIdx> <upperLimit1> <catName1> <catIdx1> ...`
  - `<catConfigName> = [<finalName>] RooThresholdCategory <catTitle> noIdx <inputVar> <defaultCatName> <upperLimit1> <catName1> ...`

  In the first form, `useIdx` is set, all the indices for threshold category types should be given explicitly. In the second form, `noIdx` is set, all the indices are set to default values.
- Syntax to create `RooSuperCategory`
  - `<catConfigName> = [<finalName>] RooSuperCategory <catTitle> <inputCat1> <inputCat2> ...`

  It creates `RooSuperCategory` based on input cats. All the indices are set to default values.
- Syntax to create `RooStringVar`
  - `<varConfigName> = [<finalName>] RooStringVar <varTitle> ["<val>"]`

  `"<val>"` is the optional initial string for the variable. If not specified, string "" (empty string) will be used.
- Syntax to create `RooFormulaVar`

- `<varConfigName> = [<finalName>] RooFormulaVar <formulaString> [<depVar0> <depVar1> ...] [<min>] [<max>]`

The title field here is actually the formula string. Depending on the actual form of the formula, there could be some variables following `<formulaString>`. Those `<depVarX>`s can be any `RooAbsReal` variables defined within the same configuration section or elsewhere. If the `RooFormulaVar` can have 'fundamental' `RooRealVar`, for example, as in dataset definition section, optional variable range limits can be specified.

2.1.2 Dataset Input Section

The container of all datasets, an object of class rarDatasets (`#item_rarDatasets`), is instantiated by `main` function of the program, and its configuration section, `[Dataset Input]`, can be changed by command line option, `-D "<data input section name>"`.

- `dsdSec = <dataset definition section>`
  This configuration specifies the name of the dataset definition section. If not specified, the default name is `[Dataset Definition]`.
- `Datasets = <data01> ... <dataN>`
- `<data01> = <dataset creation string>`
- `...`
- `<dataN> = <dataset creation string>`
  `Datasets` specifies all the datasets need to be configured. This configuration is required for this configuration section. You can also put this config, `Datasets`, into individual action sections, in which case it will override the one specified here. This is useful when you want to have different datasets for different actions.

  Configs named `<data01>`, `...`, `<dataN>`, are the definitions of all the datasets declared with `Datasets`. In fact, each declared dataset should have its definition. There are five ways to create a dataset, all of which are implemented in rarConfig::createDataSet (`#item_rarConfig`).

  1. Create dataset from ascii data file
     - `<dsName> = ascii "<dataset Title>" "<dataSetFileName>" [<options>] ["<Common Path>"] [<indexCatName>]`

     The meaning of all the fields are pretty straightforward. The order of the columns in ascii file should match that of `Fields` in dataset definition section, See Section 2.1.1 [Dataset Definition], page 11. Optional `<options>` will be passed to RooDataSet::read (`http://root.cern.ch/root/html/RooDataSet.html#RooDataSet:read`) and value of '`Q`' will suppress warning messages. If `"<Common Path>"` is specified, `"<dataSetFileName>"` can be a string of several dataset file names separated by comma. I have not found how to use `<indexCatName>`, but it is there just in case somebody knows howto and wants to use it.
  2. Create dataset from root data file
     - `<dsName> = root "<dataset Title>" "<dataSetFileName>" "<ntupleID>" ["<optional Cuts>"]`

     The meaning of all the fields are pretty straightforward.

3. Create dataset by addition
   - `<dsName> = add "<dataset Title>" <dsSrcName1> <#Evt1> [<dsSrcName2> <#Evt2> ...]`

   This method will extract randomly specific number of events (`<#Evt1>`, `<#Evt2>`, ...) from datasets, `<dsSrcName1>`, `<dsSrcName2>`, ..., which have already been configured. `<#EvtX>` can be any number greater than or equal to 1, which means `<#EvtX>` events (no more than the number of entries in the dataset) will be selected randomly; or it can be a number between 0 and 1, which means `<#EvtX>*<datasetEntries>` events will be selected; or it can be 0, which means the full dataset will be selected.
4. Create dataset by reduction
   - `<dsName> = reduce "<dataset Title>" <dsSrcName> "<cuts>"`

   The new dataset `<dsName>` will be created from dataset `<dsSrcName>` with `"<cuts>"` applied.
5. Create `RooDataHist`
   - `<dsName> = hist "<dataset Title>" <dsSrcName>`

   The new dataset `<dsName>` will be created from dataset `<dsSrcName>` as `RooDataHist` (`http://root.cern.ch/root/html/RooDataHist.html`)

- `setWeightVar = <no|[varName] dsName11 ... varName1 dsName21 ... varName2>`
  This optional configuration specifies if to use weight in datasets. If not specified, the default is `no`, not to use weight. If the first optional token is a `RooRealVar` in datasets, it will be the default weight variable unless further tokens give more specifications. You can use the remaining tokens to give more specific instructions. The format is dataset names followed by weight variable name. If the first token (optional) is not weight variable name, only those datasets specified explicitly are weighted.
- `tabulateDatasets = <no|yes>`
  This optional configuration specifies if to tabulate datasets. If it is set to `yes` (default `no`), a 1D table will be printed out against each category for each dataset.
- `computeCorrelations = <yes|no|dataSetName1 dataSetName2 ...>`
  This optional configuration specifies if to compute correlation matrix for datasets. A correlation table will be printed out for each dataset unless it is set to `no` (default `yes`). Or you can explicitly give a list of dataset names.
- `ub_<datasetName> = <ubToken1> <ubToken2>...`
  Any fitting action except `pdfFit` and `toyStudy` on a dataset requires that dataset to be "unblinded", which means this configuration is set to valid "tokens". The token will be given when you try to do the fitting action. Please follow the instruction given by the fitter. Unblinding dataset does not necessarily unblind your results, as long as you use RooUnblindPrecision or RooUnblindOffset and the state is set to blind, those variables they try to blind still remain blind.

A sample configuration section for dataset input is shown below:

```
[Dataset Input]
// Specify dataset definition section
```

```
dsdSec = Dataset Definition

// Specify datasets to be defined
Datasets = sigMC bkgMC onData gsbData desbData simData

// Definition for each dataset
sigMC    = ascii "sig MC Data" "omega/dat/omegaks_SIGMC.text" Q // quiet mode
bkgMC    = ascii "tot bkg MC Data" "omega/dat/omegaks_BKGMC.text" Q
onData   = ascii "onpeak Data" "omega/dat/omegaks_ONPEAK.text" Q
gsbData  = reduce "gsd Data" onData "mes<5.27"
desbData = reduce "de sb Data" onData "(de<-.1)||(de>.1)"
simData  = add "simulated Data" sigMC 0.0113 bkgMC 6000
```

2.2 PDF Definition Configuration

All PDFs are created from top to bottom with the `rarMLFitter` object as starting point, and each PDF has its own configuration section. The name of PDF is global which means you can not create a PDF with the same name twice, and the actual PDF creator, rarConfig::createPdf (`#item_rarConfig`), will return the pointer to the already created PDF instead.

If the name of a RooRarFit PDF is `myPdfName`, the configuration section for that PDF is `[myPdfName Config]`, and the name of the actual RooFit PDF is `the_myPdfName`.

In the following sub-sections, we will list configurations for each RooRarFit PDF. Unless explicitly stated, most of these configurations are NOT mandatory. If the configurations are to define default observables of RooRarFit PDF, those configurations are usually mandatory.

2.2.1 Configs Common to All RooRarFit PDFs

There are some configs in PDF config section which are common to all RooRarFit PDFs.

- `configStr = <pdfType> ["<pdf Title>"]`
  This config specifies the type of the PDF and also gives optional PDF title. Every PDF, except `rarMLFitter`, which is created by `main`, should have this config.
- `paramSec_<pdfType> = "<param Config Section>"`
- `paramSec = "<param Config Section>"`
  With these two configs, any other configs except `configStr` can be put into another config section specified. `paramSec_<pdfType>` has higher priority than `paramSec`. This is a convenient feature if you want to reuse configs of other PDF config section.
- `xtraGenerators = <generator1> [<generator2> ...]`
  This config specifies names of PDF for observables which usually have no distribution information in fit models. PDFs specified here will be used to generate those observables.
- `protDataVars = [<obs1> ...]`
  This config specifies names of observables which usually have no distribution information in PDF models, so their distributions are given by prototype dataset in toy studies. It is advisable to have such observables added here in their own PDF section instead of action section, so that your `protDataVars` can change dynamically. The name of

observable can be local name if it is observable in this PDF, or it should be that in datasets.

- `conditionalObs = [<obs1> ...]`
  This config specifies names of observables which usually have no distribution information in PDF models, *and should be taken out of normalization for fit.* It is advisable to have such observables added here in their own PDF section instead of action section, so that your `conditionalObs` can change dynamically. The name of observable can be local name if it is observable in this PDF, or it should be that in datasets.

  Please use config `protDataVars` if possible because fit with this config (`conditionalObs`) is very slow.

- `protDataEVars = [<obs1> ...]`
  This config specifies names of observables the distributions of which you want to get from prototype dataset for *E*mbedded events in toy studies. For example, the signal source data are usually from signal MC, but you certainly want the MC/realData bit in the embedded event set to real data as in your prototype dataset. It is advisable to have such observables added here in their own PDF section instead of action section, so that your protDataEVars can change dynamically. The name of observable can be local name if it is observable in this PDF, or it should be that in datasets.

  Please notice you do NOT need this config (`protDataEVars`) for most observables even if you do not have PDF to describe the distributions of them, in which case you should use config `protDataVars`. `protDataEVars` is for special case to embed events so that category/tag bit will be set to the right value even the sources are not from the right ones, for example, embed events from MC sample, and you have a category in the fit that requires real data type, then you have to replace the category type of MC with type real data.

- `fitData = <datasetName> ["<optional cut string>"]`
  This config specifies the default dataset for this PDF to do pdfFit, or other relevant operation. When not specified, the default dataset will be set to that of the PDF which creates this PDF. You can give this config an optional string as the second token, which will be applied to the dataset for additional cuts.

- `xtraPdfs = <xtraPdfName1> [<xtraPdfName2> ...]`
  This config specifies extra PDFs associated with this RooRarFit PDF. Extra PDFs are created for purposes like getting parameters for other PDFs in final PDF, etc. `<xtraPdfName1>`, ..., are the names of those extra RooRarFit PDFs to be created.

- `pdfFit = <yes|no|simFit|simFitOnly>`
  Do pdfFit in pdfFit action for this PDF (default `yes`). If this config is set to `simFit` and the final PDF model is `Simultaneous` PDF, `RooSimultaneous` PDF built for this PDF will also be fit to the default dataset. (See example in KsKsKl.config (`http://rarfit.sourceforge.net/Sample_configs/KsKsKl.config`)). If this config is set to `simFitOnly`, the prototype PDF will not be fitted.

- `firstFitOnly = <yes|no>`
  pdfFit action is done for the first time only as the PDF is created (default). If set to `no`, pdfFit will be done every time it is referred to and results from the last fit are effective.

- `fitRange_<obsName> = <Min> <Max>`
  It sets **pdfFit** ranges for observable `<obsName>`. If not specified, the range will be set to the full ranges when it is created.
- `pdfPlot = <yes|no>`
  Do pdfPlot in pdfFit action for this PDF (default `yes`). For how to disable plotting for one variable in a PDF, see next item.
- `plotBins_<obsName> = <nBins>`
  It sets plot bins for observable `<obsName>`. If not specified, the number of plot bins is set to the value when the observable is created (`B(<nBins>)`); if set to -1, the plot of this var is disabled.
- `plotRange_<obsName> = <Min> <Max>`
  It sets plot ranges for observable `<obsName>`. If not specified, the range will be set to the full ranges when it is created.
- `projWData_<obsName> = <no|yes|dataName>`
  It specifies reference dataset of observable `<obsName>` for its PDF plotting. The default is `no`, no reference dataset, which is true for most of PDF plotting. For some type of PDF, however, you need to set this reference dataset, because that PDF does not have the distribution information of other observables in it. For example, in *dt* PDF for background, there is no distribution information for *dtErr* usually, so when it is integrated over *dtErr* to get the plot of *dt*, the fitter will assume flat distribution of *dtErr* and then you will not get the right plot. If you set the config to `yes`, the default dataset for the PDF will be used, or you can explicitly give a dataset name to the config.
- `compsOnPlot = <no|yes>`
  This config controls when the RooFit PDF is composite, if its component PDF need to be plotted. The default is `no` for rarBasePdf, but sub RooRarFit PDFs can override it, for example in rarTwoGauss, the default is `yes`. Anyway, you can always set desirable value by yourself using this config.
- `compsDataOnPlot = <no|yes|refPdf>`
  This config specifies when the RooFit PDF is composite, and `compsOnPlot` is set to `yes`, if the data points for each component are plotted (default `no`). If it is set to `yes`, this PDF itself will be used as reference to plot each component's data points; or you can explicitly give a reference PDF name to the config.
- `paramsOnPlot = <yes|no>`
  Params on PDF plot, default yes.
- `chi2OnPlot = <yes|no|dof|nbin>`
  Chi-square on PDF plot, default `yes`. The number showing is chi square over DOF by default, and you can set this config to `nbin` so chi square over number of bins will be displayed.
- `plotWCat_<obs> = <no|CatName1...>`
- `plotWCat = <no|CatName1...>`
  pdfPlot will also be done for each type of the cats specified here (default `no`).
- `prePdfFix = <paramName1> [va1] [<paramName2> [val2] ...]`
  This config specifies parameters to be fixed before pdfFit. It is useful when those parameters have already been determined and you do not want them to float in pdfFit for

THIS PDF config section only. It will not change the attributions of those parameters they will be floated or fixed in the parameter files, or in mlFit, etc. The name should be full name if the parameter is not defined in the same section; the name can also be name of the PDF itself, or its component PDFs. If the name is a PDF name, all the *direct* parameters of that PDF will be included. If the name is parameter name and the next argument is a number, that parameter will be set to that value and fixed. After pdfFit, parameter values will remain the same as you specify here, and their constant properties will be restored.

Please notice that if a PDF name is specified, only its *direct* parameters will be included, ie, if it is composite PDF, you need to specify its components' PDF names or parameters to include parameters from its components, otherwise parameters from component PDFs will *NOT* be fixed.

- `prePdfFloat = <paramName1> [<paramName2> ...]`
  This config specifies parameters to be floated before pdfFit. It is usually unnecessary because parameters for PDF are usually defined as floating ones, but if it is more convenient to declare parameters as constant, and if it is needed, use this config to float those parameters in PDF fit. The name should be full name if the parameter is not defined in the same section; the name can also be name of the PDF itself, or its component PDFs. If the name is a PDF name, all the direct parameters of that PDF will be included.

  Both `prePdfFix` and `prePdfFloat` affect *ONLY* the PDF where they are immediately at. If they are in a sub-PDF and this PDF is part of a total PDF, you need to have `prePdfFix` or `prePdfFloat` in the total PDF if you want to fix/float those parameters in the total PDF.
- `postPdfFloat = <paramName1> [<paramName2> ...]`
  This config specifies parameters to be floated after pdfFit. The name should be full name if it is not defined in the same section, and it can be the name of the PDF itself. If the name is a PDF name, all the direct parameters of that PDF will be included.

  This config can be put into action section so it works at per action basis.
- `preMLFix = <paramName1> [<paramName2> ...]`
  This config specifies parameters to be fixed before all actions except pdfFit. The name should be full name if it is not defined in the same section, and it can be the name of the PDF itself. If the name is a PDF name, all the direct parameters of that PDF will be included. It is used to undo `postPdfFloat`.

  This config can be put into action section so it works at per action basis.
- `preMLFloat = <paramName1> [<paramName2> ...]`
  This config specifies parameters to be floated before all actions except pdfFit. The name should be full name if it is not defined in the same section, and it can be the name of the PDF itself. If the name is a PDF name, all the direct parameters of that PDF will be included. It is used to expand `postPdfFloat`.

  This config can be put into action section so it works at per action basis.
- `Ignored = <paramName1> [<paramName2> ...]`
  This config specifies parameters to be ignored after actions, which means those parameters will not be output to parameter files. This is useful when you do not want the fitter to override parameter initial values. The name should be full name if it is not

defined in the same section, and it can be the name of the PDF itself. If the name is a PDF name, all the direct parameters of that PDF will be ignored.

2.2.2 rarMLFitter Configs

The final mlFitter, an object of class `rarMLFitter`, is instantiated by `main` function of the program, and its config section, `[mlFitter Config]`, can be changed by command line option, `-C "<mlFitter config section>"`. This is the master PDF config section, the configs of which might affect all other PDF sections.

- `fullNameSchema = <prefix|suffix|self>`
  This config specifies how the full name of a variable, if not given, is generated. This config affects all RooRarFit PDFs. The default is `prefix` which creates full name by prepending name of the creating pdf to the variable config name; `suffix` creates full name by appending the pdf name to the variable config name; `self` sets the full name as the variable config name. `self` should not be used for PDF, and it is the default method for `rarDatasetDef` because any variable created in dataset definition section should be unique by itself.

  This config is fixed to `prefix` and no change is permitted.
- `outParamOrder = <ascend|descend|unsorted>`
  It controls how the final output params are sorted. The order, based on param names, could be ascending (`ascend`, default), descending (`descend`), or in the original order as they are created (`unsorted`).
- `useNumCPU = <1-8>`
  The number of CPUs (cores) to be used in the ML fits. The fitter will try to parallelise the fit over the available cores. (default `1`).
- `Comps = <compPdfName1> [<compPdfName2> ...]`
  This config specifies prototype pdf models for mlFitter. If the final mlFitter is not simultaneous pdf, only the first component is used. This config is mandatory for any pdf inherited from `rarCompBase`.
- `simultaneousFit = <no|yes>`
  This config specifies if the final mlFitter is SimPdf built by `RooSimPdfBuilder` or not (default `no`). If set to `yes`, you need to have configs for `RooSimPdfBuilder` to get the SimPdf built. Detailed info on how to use these configs can be found from `RooSimPdfBuilder` online document.

  Essentially, the splitting can be done manually by fitting on to separate sub-datasets for individual categories. So for each category, the yields, `nSig`, `nBkg`, etc. need to be split, which means for `n` number of category types, there should be `n` sub-yields. To get the total yield, just sum up all these values. It is then convenient to have the total yield, and `n-1` sub-yield fractions as free parameters:

  ```
  subY1=Ntot*(1-frac2...-fracn),
  subY2=Ntot*frac2,
  ...
  subYn=Ntot*fracn.
  ```

  That is the basic idea for the coeffs splitting of the extended AddPdf, `rarMLPdf`. For each category, you will need similar splitting settings.

- `yieldSplitMethod = <auto|manual>`

  If this config is set to `manual`, the user has full control over yield splitting. He can choose to split directly on yields, or to construct the yield with `RooFormulaVar`, and split the formula variables, etc. It is not recommended unless the other two methods do not meet your needs, and you know exactly what to do to achieve your goals.

  Since it is a quite complicated issue to split yield for each category, you'd better let the fitter do it for you. The other two options are for this purpose. In either case, as stated in previous item, for every splitting category, the yield variable multiplies a splitting fraction variable. For example, if `runBlock` is one of the splitting categories, for signal yield, the variable will be automatically re-defined to `nSig*fracRunBlock`, where `nSig` is the final yield, and `fracRunBlock` is the automatically created splitting fraction variable for `runBlock`. If there are four run blocks, `fracRunBlock` will be split into four variables for the four run blocks. If `nSig` is the signal yield name, `runBlock` is run block category name, and four category types are `run1`, `run2`, `run3`, and `run4`, the name of `fracRunBlock` will be `Frac_nSig_runBlock`, and the four split variables are

  ```
  Frac_nSig_runBlock_run1,
  Frac_nSig_runBlock_run2,
  Frac_nSig_runBlock_run3,
  Frac_nSig_runBlock_run4.
  ```

  You can give, for example, `Frac_nSig_runBlock`, a different name, or even assign it to an existing var, by explicitly giving its definition in this section, like

  ```
  Frac_nSig_runBlock = e_run_sig 1 C L(0 - 1)
  ```

  and the four split variables are then

  ```
  e_run_sig_run1,
  e_run_sig_run2,
  e_run_sig_run3,
  e_run_sig_run4.
  ```

  If `yieldSplitMethod` is set to `auto` (default), `RooFit` will impose unity constraint on the fractions if you specify the cat type in square bracket for those fractions. For each yield and splitting category pair, you have to specify the non-free cat type in square bracket; otherwise the fitter will abort once it detects such situation.

- `fracRule = <catGroup1>...`
- `fracRule_<yieldName> = <catGroup1>...`
  By default, each yield has separate splitting fraction for each category, but in some cases, for example, if fractions for two categories are correlated, it might be desirable to split more than one category together. For example, if you want to split (double-split) runBlock and tagCat together for `nSig`, split tagCat and XCat together for `nChmls`, you can have

  ```
  fracRule_nSig   = "runBlock tagCat" XCat
  fracRule_nChmls = runBlock "tagCat XCat"
  ```

  For `runBlock tagCat` combined splitting, you will have `Frac_nSig_runBlock_tagCat` as the splitting fraction var, and the split vars will be `Frac_nSig_runBlock_tagCat_{Run1;04T0}`, etc.

- `fracSrc = <datasetName>`
- `fracSrc_<yieldName> = <datasetName>`
- `fracSrc_<yieldName>_<catName> = <datasetName>`
  These configs specify, when `yieldSplitMethod` is `auto`, the source for splitting fractions. The default source is `fitData`. `fracSrc`, if specified, is for all yields, and all cats; `fracSrc_<yieldName>`, if specified, is for all cats of yield `yieldName`; `fracSrc_<yieldName>_<catName>`, if specified, is for cat `catName` of yield `yieldName`. Those configs can be put into actions so they can be on per action basis.
- `splitSpecials = [fullNamed] <specialVar1> [<specialVar2>...]`
- `<specialVar1> = AbsReal Def`
- `<specialVar2> = AbsReal Def`
- `...`
  This config specifies names and definitions of specializations for splitting. If you want to set a specific value for a split variable, or to give it a different definition than `RooRealVar`, you can add that split variable into this config. If the first arg is `fullNamed`, all the following configs are supposed to be full names.

After pdf has been created from top to bottom, action section will be invoked to finish fitting jobs. Intermediate fitting results will be saved for later uses. It is crucial for each action to get desired initial values from the right param files. So the fitter is designed to have param file name for each fitter, action, etc., generated automatically. User can specify param file creation rules and refer to the destination param file by the rules. For each action, there are configs to specify IO param files. You can also have those param file configs in the master section, which will override those in action section.

- `postPdfWriteParams = <notSet|no|yes|paramFileID>`
  If specified, it will override `postPdfWriteParams` of pdfFit action for this master section.
- `preMLReadParams = <notSet|no|yes|paramFileID>`
  If specified, it will override `preMLReadParams` of mlFit action for this master section.
- `postMLWriteParams = <notSet|no|yes|paramFileID>`
  If specified, it will override `postMLWriteParams` of mlFit action for this master section.
- `preToyReadParams = <notSet|no|yes|paramFileID>`
  If specified, it will override `preToyReadParams` of toyStudy action for this master section.
- `toyDataFilePrefix = <notSet|toySampleNameID>`
  If specified, it will override `toyDataFilePrefix` of toyStudy action for this master section.
- `preProjPlotReadParams = <notSet|no|yes|paramFileID>`
  If specified, it will override `preProjPlotReadParams` of projPlot action for this master section.
- `preContourPlotReadParams = <notSet|no|yes|paramFileID>`
  If specified, it will override `preContourPlotReadParams` of contourPlot action for this master section.
- `preSPlotReadParams = <notSet|no|yes|paramFileID>`
  If specified, it will override `preSPlotReadParams` of sPlot action for this master section.

- `<paramFileID>` or `<toySampleNameID>` specifications
  It can have one or more of the following pairs to specify the param/toysample.
  - `F "<configFileName>"`

    The naming rules depending on config file name will be changed to the value specified here. The default value is the name of this job's config file.
  - `D "<datasetInputSecName>"`
    The naming rules depending on dataset input section name will be changed to the value specified here. The default value is the name of this job's dataset input section.
  - `C "<masterPdfSecName>"`
    The naming rules depending on master pdf config section name will be changed to the value specified here. The default value is the name of this job's master pdf config section.
  - `A "<actionTypeName>"`
    The naming rules depending on action type name will be changed to the value specified here. The action types include `pdfFit` and `mlFit` for intermediate param file. You can use arbitrary name, and you just need to refer to the same name accordingly.
  - `N "<conceptualParamFileName>"`
    The naming rules depending on conceptual param file name will be changed to the value specified here. There is no default abstract name for intermediate param file. You can use arbitrary name here, and you just need to refer to the same name accordingly.
- `mlFitData = <datasetName> ["<optional cut string>"]`
  If specified, it will override `mlFitData` of mlFit action for this master section.
- `projPlotData_<obs> = <datasetName> ["<optional cut string>"]`
- `projPlotData = <datasetName> ["<optional cut string>"]`
  If specified, it will override `projPlotData` or `projPlotData_<obs>` of projPlot action for this master section.
- `scanPlotData = <datasetName>`
  If specified, it will override `scanPlotData` of scanPlot action for this master section.
- `sPlotData_<obs> = <datasetName> ["<optional cut string>"]`
- `sPlotData = <datasetName> ["<optional cut string>"]`
  If specified, it will override `sPlotData` or `sPlotData_<obs>` of sPlot action for this master section.

An example is shown below:

```
[mlFitter Config]
Comps = yieldModel
fitData = onData

simultaneousFit = yes

// SimPdfBuilder configs
```

```
physModels = the_yieldModel // Please remember the RooFit pdf is created with a
                            // "the_" before the name of RooRarFit PDF object.
splitCats = kltype tagCat
protDataVars = kltype tagCat
the_yieldModel  = kltype : sigma1_deSigTriGauss      \\
                  kltype,tagCat : sigma2_deSigTriGauss, P01_esNN2SigPoly \\
                  kltype : Frac_nSig_kltype[ifr], Frac_nBkg_kltype[ifr] \\
                  tagCat : Frac_nSig_tagCat[04T0], Frac_nBkg_tagCat[04T0]
```

2.2.3 rarMLPdf Configs

`rarMLPdf` is an extended AddPdf, which serves as prototype pdf for final mlFitter if the final mlFitter is SimPdf, or it will be the final mlFit model if the final mlFitter is not SimPdf.

- `configStr = MLPdf ["<pdf Title>"]`
  This config specifies the pdf type is `MLPdf`. This config is required to have this pdf configured as `rarMLPdf`.
- `Comps = <compPdfName1> [<compPdfName2> ... <compPdfNameN>]`
  This config specifies components for `rarMLPdf`. The components usually consist of one or more signal parts, one continuum background, any *BB* backgrounds, etc. This config is mandatory for any pdf inherited from `rarCompBase`.
- `Coeffs = [fullNamed] <coeff1> [<coeff2> ... <coeffN>]`
  This config specifies coefficients for all its components. Since this pdf is an extended AddPdf, the number of coefficients should match that of components. If the first arg is `fullNamed`, all the following configs are supposed to be full names. This config is mandatory for any pdf inherited from `rarAddPdf`.
- `<coeff1> = AbsReal Def`
- `...`
- `<coeffN> = AbsReal Def`
  All the coeffs can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[yieldModel Config]
configStr=MLPdf "ml model"
Comps=SigPdf BkgPdf ChlsPdf
Coeffs=nSig nBkg nChls
nBkg =nBkg  6000 L(  0 - 10000)
nSig =nSig   156 L(-10 -   500)
nChls=nChls 1300 L(  0 -  3000)
```

2.2.4 rarMultPdf Configs

`rarMultPdf` is a wrapper of `RooGenericPdf` to build a special pdf to deal with different fit ranges for multiple sub mode in simPdf fit.

- `configStr = MultPdf ["<pdf Title>"]`
  This configuration specifies the pdf type is `MultPdf`. This configuration is required to have this pdf configured as `rarMultPdf`.

- `Comps = <compPdfName1> [<compPdfName2> ... <compPdfNameN>]`
  This configuration specifies components for `rarMultPdf`, and is mandatory for any pdf inherited from `rarCompBase`.

An example is shown below:

```
[mESigS Config]
configStr = MultPdf
Comps = mESig mEStep

[mESig Config]
configStr = TwoGauss
x = mEta
meanC =  0.540 L(0.52 - 0.58)
meanT =  0.530 L(0.52 - 0.58)
sigmaC = 0.012 L(0.0  - 0.02)
sigmaT = 0.030 L(0.0  - 0.1)
fracC =  0.65   L(0.0  - 1.0)

[mEStep Config]
configStr = Generic
x = mEta
formula = "@1 <= @0 && @0 <= @2" x mEStepLo mEStepHi
mEStepLo = mEStep_mEStepLo 0.485 C L( 0 - 1.0)
mEStepHi = mEStep_mEStepHi 0.605 C L( 0 - 1.0)
```

2.2.5 rarProd Configs

`rarProd` is a wrapper of `RooProdPdf` to build composite pdf as a product of PDFs.

- `configStr = ProdPdf ["<pdf Title>"]`
  This config specifies the pdf type is `ProdPdf`. This config is required to have this pdf configured as `rarProd`.
- `Comps = <compPdfName1> [<compPdfName2> ... <compPdfNameN>]`
  This config specifies components for `rarProd`, and is mandatory for any pdf inherited from `rarCompBase`.
- `CondPdfs = [condPdf1] ...`
  This optional config specifies conditional PDFs for `rarProd`. The normalization of PDFs sepcified here is on observables in config `CondObss` only.
- `CondObss= [condObs1] ...`
- `CondObss_<condPdf> = [condObs1] ...`
  This optional config specifies conditional observables of conditional PDFs in `CondPdfs`. Only observables listed here are included in the normalization calculation for conditional PDFs. Each conditional PDF should have its own `CondObss_<condPdf>` config. If config `CondObss` exits, the observables specified will be added to all condtional PDFs in current product PDF.
- `ndFit = <no|yes>`
  Do `ndFit` in `pdfFit` action if set to `yes` (default `no`). `ndFit` (n-dimensional fit) will fit product pdf all together instead of fit individual components.

For rarProd with conditional PDFs, ndFit is set to yes.

An example is shown below:

```
[SigPdf Config]
configStr = ProdPdf "Signal Pdf"
Comps = deSig mesSig fisherSig mOmegaSig heliSig
fitData = sigMC
```

An example of multivariate PDF is shown below:

```
// The 2D PDF is defined in terms of P(h) and conditional PDF P(m|h)
// P(m,h)=P(m|h)*P(h)
[hm2D Config]
configStr = ProdPdf "P(m,h)"
Comps = hPdf mPdf
CondPdfs = mPdf
CondObss = m
fitData = sigMC
```

2.2.6 rarAdd Configs

rarAdd is a wrapper of RooAddPdf/RooAddModel to build composite pdf as a sum of PDFs. There are N-1 free coefficients with N components, or N coefficients if the composite pdf is extended. RooAddModel can only be non-extended pdf.

- configStr = AddPdf ["<pdf Title>"]
- configStr = AddModel ["<pdf Title>"]
  This config specifies the pdf type is AddPdf/AddModel. This config is required to have this pdf configured as rarAdd.
- Comps = <compPdfName1> [<compPdfName2> ... <compPdfNameN>]
  This config specifies components for rarAdd, and is mandatory for any pdf inherited from rarCompBase.
- Coeffs = [fullNamed] <coeff1> [<coeff2> ... <coeffM>]
  This config specifies coefficients for all its components. If M is equal to N, it is extended, or M should be N-1, which means the pdf created is not extended. If the first arg is fullNamed, all the following configs are supposed to be full names. This config is mandatory.
- <coeff1> = AbsReal Def
- ...
- <coeffM> = AbsReal Def
  All the coeffs can be defined as RooRealVar or RooFormulaVar. See createVar (#item_createVar) for more info on how to create those variables.

An example is shown below:

```
[fisBkg Config]
configStr = AddPdf "Add two Pdfs"
Comps  = fisBkgC fisBkgT
Coeffs = fracC
fracC  = T "f_{C}" 0.9 L(0 - 1.)
```

2.2.7 rarSimPdf Configs

`rarSimPdf` uses `RooSimPdfBuilder` to build `RooSimultaneous` Pdf. *This pdf can not be used to build ML fitter model!*

- `configStr = Simultaneous ["<pdf Title>"]`
  This config specifies the pdf type is `Simultaneous`. This config is required to have this pdf configured as `rarSimPdf`.
- `Comps = <compPdfName1> [<compPdfName2> ... <compPdfNameN>]`
  This config specifies components for `rarSimPdf`, and is mandatory for any pdf inherited from `rarCompBase`.
- `physModels = <RooSimPdfBuilder config string>`
  This config specifies physics models.
- `splitCats = <cat1>...`
  This config specifies splitting cats.
- `the_<compPdfName1> = <splitting rules>`
  This config specifies splitting rules for each model.

2.2.8 rarArgusBG Configs

`rarArgusBG` is a wrapper of `RooArgusBG` to build ArgusBG PDF. This distribution was first used by the Argus experiment to descibe the combinatorial background.

- `configStr = ArgusBG ["<pdf Title>"]`
  This config specifies the pdf type is `ArgusBG`. This config is required to have this pdf configured as `rarArgusBG`.
- `x = AbsReal Def`
- `max = AbsReal Def`
- `c = AbsReal Def`
- `pow = AbsReal Def`
  `x` is the default observable of the pdf. `max` is the end point of the pdf, usually a constant. `c` is the slope parameter of the pdf. `pow` is the power of the multiplying term (normally 1/2). All the variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[mesBkg Config]
fitData = desbData
configStr = ArgusBG
x = mes
max = 5.29 C
c = -20 L(-80 - -.1)
pow = 0.5 L(-3 - 3)
postPdfFloat = c
```

2.2.9 rarBallack Configs

`rarBallack` implements the Ballack function.

- `configStr = Ballack ["<pdf Title>"]`
  This config specifies the pdf type is `Ballack`. This config is required to have this pdf configured as `rarBallack`.
- `x = AbsReal Def`
- `mean = AbsReal Def`
- `width = AbsReal Def`
- `tail = AbsReal Def`
- `alpha = AbsReal Def`
- `n = AbsReal Def`

`x` is the default observable of the pdf. `mean` is the mean. `width` is the width. `tail` is the tail. `alpha` is the expoential. `n` is the power.

All the variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[mSig Config]
configStr = Ballack
x     = de
mean  = 0 L (-1 - 1)
sigma = 0.003 L(0 - .01)
alpha = -2 L(-5 - 0)
n     = 1 L(0 - 10)
```

2.2.10 rarBinned Configs

`rarBinned` implements a binned function.

- `configStr = Binned ["<pdf Title>"]`
  This config specifies the pdf type is `Binned`. This config is required to have this pdf configured as `rarBinned`.
- `x = AbsReal Def`
- `nbins = AbsReal Def`
- `limits = AbsReal Def`

`x` is the default observable of the pdf. `nbins` is the number of bins of the binned function. `limits` is a list on nbins+1 setting the bounds of those bins. `H00...` are the nbins+1 free parameters of the binned function. All the variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[mSig Config]
configStr = Binned
x = de
nbins  = 3
limits = 0.0 2.0 4.0 5.0
H00    = 5 L (0.0 - 1.0)
```

2.2.11 rarCruijff Configs

`rarCruijff` implements a Cruijff function.This distribution is similar to that of a Gaussian with different widths above and below the mean together with a low and high exponential tail.

- `configStr = Cruijff ["<pdf Title>"]`
  This config specifies the pdf type is `Cruijff`. This config is required to have this pdf configured as `rarCruijff`.
- `x = AbsReal Def`
- `mean = AbsReal Def`
- `sigmaL = AbsReal Def`
- `sigmaR = AbsReal Def`
- `alphaL = AbsReal Def`
- `alphaR = AbsReal Def`

`x` is the default observable of the pdf. `mean` is the mean. `sigmaL` is the left/low width. `sigmaR` is the right/high width. `alphaL` is the left/low exponential tail. `alphaR` is the right/high exponential tail. All the variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[cruijffSig Config]
configStr = Cruijff
x     = de
mean     = 0 L (-1 - 1)
sigmaL = 0.003 L(0 - .01)
sigmaR = 0.004 L(0 - .01)
alphaL = 0.01 L(0 - 1)
alphaR = 0.02 L(0 - 1)
```

2.2.12 rarDecay Configs

`rarDecay` is a wrapper of `RooBCPGenDecay/RooBDecay/RooDecay` to build BCPGenDecay / BDecay / Decay Model. These are single or double sided decay functions that can be convolved with a resolution model such as GaussModel. `Decay` is an exponential decay; `BDecay` models the decay of the B meson; `BMixDecay` includes mixing the decay; `BCPGenDecay` includes CP violation in the decay.

- `configStr = BCPGenDecay ["<pdf Title>"]`
- `configStr = BDecay ["<pdf Title>"]`
- `configStr = Decay ["<pdf Title>"]`
  This config specifies the pdf type is `BCPGenDecay`, `BDecay` or `Decay`. This config is required to have this pdf configured as `rarDecay`.
- `x = RealVar Def`
- `tau = AbsReal Def`
- `model = <resolutionModelName>`

- decayType = <typeName>
- dm = AbsReal Def
- dgamma = AbsReal Def
- tag = AbsCat Def
- w = AbsReal Def
- dw = AbsReal Def
- mu = AbsReal Def
- S = AbsReal Def
- C = AbsReal Def
- blindStatus = <blind|unblind>
- blindString = <unique blind string>
- blindValues = <Cvalue> <Cscale> <Svalue> <Sscale>

x is the default observable of the pdf. No derived dependent is allowed in rarDecay so please make sure x is observable defined in datasets. tau is the average *B0* lifetime. model is the resolution model. decayType can be SingleSided, DoubleSided (default), Flipped.

For BCPGenDecay and BDecay, dm is the mixing frequency; dgamma is the difference in $B0/B0$bar width; tag is the flavor tag category; w is the mistag rate; dw is the mistag rate difference between $B0/B0$bar; mu is the difference in tagging efficiency between $B0/B0$bar; S is *CP* sine parameter, and C *CP* cosine parameter, the CP asymmetry parameters The default blindStatus is blind, to unblind, blindStatus must be set to unblind. blindString must be set to a unique string for blinding. blindValues specifies blinding values and scales for C and S, (default .2 .2 .2 .2).

All the 'AbsReal' parameters can be defined as RooRealVar or RooFormulaVar, the 'AbsCat' parameter can be defined as any RooAbsCategory. See createVar (#item_createVar) for more info on how to create those variables.

An example is shown below:

```
[dtSig Config]
configStr = BCPGenDecay "CPV signal"
projWData_dt = yes
//pdfFit = no
//pdfPlot = no
x = dt
model = sigResModel
tag = tagFlav                          // tagging category
tau = dtSigTau  1.530 C L(0.5 - 2.5) // B0 lifetime
dm  = dtSigDm   0.507 C L(0.2 - 1.0) // B0 mixing frequency
w   = dtSigAmtr 0.2  C L(0.0 - 0.5)  // mistag rate
dw  = dtSigDmtr 0.02 C L(-2. - 2.0)  // B0/B0bar mistag rate difference
mu  = dtSigMu   0.0  C L(-2. - 2.0)  // B0/B0bar tagging efficiency difference
C = 0.0 L(-3 - 3)                    // direct CP parameter
S = 0.7 L(-3 - 3)                    // indirect CP parameter
blindStatus = unblind
```

```
blindString = Physicists do it vivace
blindValues = .2 .2 .2 .2
postPdfFloat = C S

[sigResModel Config]
configStr = TriGaussModel "CPV signal resolution model"
x = dt
meanC  = dtSigBiasC "BiasC"  -0.16 C L(-5 - +5)
sigmaC = dtSigScfaC "ScaleC"  0.732813 C L(.5 - 5)
meanT  = dtSigBiasT "BiasT"  -1.1140 C  L(-5 - 0)
sigmaT = dtSigScfaT "ScaleT"  3.00  C L(.5 - 10)
meanO  = dtSigBiasO "BiasO"   0. C
sigmaO = dtSigScfaO "ScaleO"  8. C
fracC  = dtSigFracC "fC"      0.8888 C L(.5 - 1)
fracO  = dtSigFracO "f0"      0.0033 C L(0. - 0.2)
msSF = dterr
protDataVars = dterr
xtraGenerators = dterrGen
```

2.2.13 rarBifurGauss Configs

`rarBifurGauss` is a wrapper of `RooBifurGauss` to build Bifurcated Gaussian PDF.

- `configStr = BifurGauss ["<pdf Title>"]`
- `configStr = BGGauss ["<pdf Title>"]`
  This config specifies the pdf type is `BifurGauss` or `BGGauss`. This config is required to have this pdf configured as `rarBifurGauss`.
- `parSymLevel = <0|1|2|3>`
- `x = AbsReal Def`
- `peak = AbsReal Def`
- `sigL = AbsReal Def`
- `sigR = AbsReal Def`
- `mean = AbsReal Def`
- `rms = AbsReal Def`
- `asym = AbsReal Def`
  If `parSymLevel` = 0, use the default BifurGauss; `mean`, `rms`, and `asym` are not required. If `parSymLevel` = 1, use lowest-order mapping; if = 2, keep O(A^2, A^3) terms also; if = 3, take `asym` to mean 3rd moment. With `parSymLevel` > 0, `mean`, `rms`, and `asym` are used; `peak`, `sigL`, and `sigR` are hard-coded to the corresponding `RooFormulaVar`. `x` is the default observable of the pdf. Pdf type `BifurGauss` and `BGGauss` are interchangeable, with the only difference being that if `parSymLevel` not specified, the default value of it for `BifurGauss` is 0, while that for `BGGauss` is 1. All the `AbsReal` variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[fisherSig Config]
```

```
configStr = BifurGauss
x    = fisher
peak = -.5 L(-2 - 2)
sigL =  .4 L(0 - 1)
sigR =  .6 L(0 - 1)

[fisherSig Config]
configStr = BGGauss
x = fisher
mean  = -.5 L(-2 - 2)
rms   =  .1 L(-0.5 - 0.5)
asym  =  .6 L(0 - 1)
```

2.2.14 rarCBShape Configs

`rarCBShape` is a wrapper of `RooCBShape` to build Crystal Ball lineshape PDF. It is a Gaussian with an exponential tail.

- `configStr = CBShape ["<pdf Title>"]`
  This config specifies the pdf type is `CBShape`. This config is required to have this pdf configured as `rarCBShape`.
- `x = AbsReal Def`
- `mean = AbsReal Def`
- `sigma = AbsReal Def`
- `alpha = AbsReal Def`
- `n = AbsReal Def`
  `x` is the default observable of the pdf. All the variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[deSig Config]
configStr = CBShape
x         = de
mean      = 0 L(-.01 - .02)
sigma     = 0.003 L(0 - .01)
alpha     = -2 L(-5 - 0)
n         = 1 L(0 - 10)
```

2.2.15 rarExp Configs

`rarExp` is a wrapper of `RooExponential` to build Exponential PDF.

- `configStr = Exp ["<pdf Title>"]`
  This config specifies the pdf type is `Exp`. This config is required to have this pdf configured as `rarExp`.
- `x = AbsReal Def`
- `c = AbsReal Def`
  `x` is the default observable of the pdf. `c` is the exponent of the pdf. All the variables

can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[esNN3Exp Config]
// the actual final pdf is exp(c*x*x)
configStr=Exp
x=RooFormulaVar "@0*@0" esNN3
c=36 L(0 - 100)
```

2.2.16 rarFlatte Configs

`rarFlatte` is a wrapper of `RooFlatte` to build a Flatte parameterisation PDF. This is typically used in the decays of the a0(980) or f0(980) where the invariant mass distribution of light scalar mesons near the K Kbar threshold. References: S.M.Flatte Phys. Rev. B63, 224 (1976); B.S.Zou and D.V.Bugg Phys Rev. D48, R3948 (1993); M.Ablikim wt al (BES collaboration), Phys. Rev. D70, 092002, (2004)

- `configStr = Flatte ["<pdf Title>"]`
  This config specifies the pdf type is `Flatte`. This config is required to have this pdf configured as `rarFlatte`.
- `x = AbsReal Def`
- `mean = AbsReal Def`
- `width = AbsReal Def`
- `g0 = AbsReal Def`
- `m0a = AbsReal Def`
- `m0b = AbsReal Def`
- `g1 = AbsReal Def`
- `m1a = AbsReal Def`
- `m1b = AbsReal Def`

  `x` is the default observable of the pdf. `mean` is the mean of the pdf. `g0` is the square of the coupling constant to the first decay channel. `m0a` and `m0b` are the masses of the two final state particles in the first channel (e.g. f0(980) -> pi+ pi-). `g1` is the square of the coupling constant to the second decay channel. `m1a` and `m1b` are the masses of the two final state particles in the second channel (e.g. f0(980) -> K+ K-). After being defined, the masses are always held constant in the fit. The default values assume that everything is defined in GeV; if using a different unit, ensure all variables are rescaled. All the `AbsReal` variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[massSig Config]
configStr = Flatte "f0(980) K*"
x = mass                    // GeV
mean  = 0.975 L(0.9 - 1.0)  // GeV
g0  = 0.1108 L(0.02 - 1.0)  // GeV
g1  = 0.4229 L(0.02 - 1.0) // GeV
```

```
m0a = 0.13957 C // pi+ mass
m0b = 0.13957 C // pi- mass
m1a = 0.49368 C // K+ mass
m1a = 0.49368 C // K- mass
```

2.2.17 rarGaussian Configs

`rarGaussian` is a wrapper of `RooGaussian/RooBreitWigner/RooLandau` to build Gaussian / Breit-Wigner / Landau PDF.

- `configStr = Gaussian ["<pdf Title>"]`
- `configStr = BreitWigner ["<pdf Title>"]`
- `configStr = Landau ["<pdf Title>"]`
  This config specifies the pdf type is `Gaussian`, `BreitWigner`, or `Landau` This config is required to have this pdf configured as `rarGaussian`.
- `x = AbsReal Def`
- `mean = AbsReal Def`
- `sigma = AbsReal Def`
- `scale = AbsReal Def`
- `shift = AbsReal Def`
  `x` is the default observable of the pdf. `mean` is the mean of the pdf. `sigma` is the sigma of the pdf. `sigma` will be scaled by the value of `scale` if specified; `mean` will be shifted by the value of `shift` if specified. All the variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[deSig Config]
configStr = BreitWigner
x = de
mean = 0 L(-.01 - 0.08)
sigma = 0.003 L(0 - .1)

[deSig Config]
configStr = Gaussian
x = de
mean = 0 L(-.01 - 0.08)
sigma = 0.003 L(0 - .1)

[dtErrSig Config]
configStr = Landau
x = dtErr
mean = .75 L(0 - 2.5)
sigma = .17 L(0 - .5)
```

2.2.18 rarGaussModel Configs

`rarGaussModel` is a wrapper of `RooGaussModel` to build Gaussian Resolution Model.

- `configStr = GaussModel ["<pdf Title>"]`
  This config specifies the pdf type is `GaussModel`. This config is required to have this pdf configured as `rarGaussModel`.
- `x = RealVar Def`
- `mean = AbsReal Def`
- `sigma = AbsReal Def`
- `msSF = AbsReal Def`
- `meanSF = AbsReal Def`
- `sigmaSF = AbsReal Def`
  `x` is the default observable of the pdf. No derived dependent is allowed in rarGaussModel so please make sure `x` is observable defined in datasets. `msSF` is the scale for `mean` and `sigma`; `meanSF` is the scale for `mean`, and `sigmaSF` for `sigma`. One can choose to specify `msSF` for both `mean` and `sigma`, or choose `meanSF` for `mean`, and/or `sigmaSF` for `sigma`, or if none is chosen, the default values are 1. All the parameters can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.
- `FlatScaleFactorIntegral = <yes|no>`
  This optional config specifies to use FlatScaleFactorIntegral or not. The default is `yes`. To disable FlatScaleFactorIntegral, this config must be set to `no`.

An example is shown below:

```
[dtSigModel Config]
configStr=GaussModel
x=dt
mean=-0.1715 L(-5 - +5)
sigma=1.0893 L(.5 - 5)
msSF=dtErr
```

2.2.19 rarGeneric Configs

`rarGeneric` is a wrapper of `RooGenericPdf` to build Generic PDF using string expression and list of variables.

- `configStr = Generic ["<pdf Title>"]`
  This config specifies the pdf type is `Generic`. This config is required to have this pdf configured as `rarGeneric`.
- `formula = "<formulaStr>" <var1> <var2> ...`
  `formulaStr` is the expression string as in `RooFormulaVar`. `<var1>`, `<var2>`, `...`, are the list of variables for the formula. All the variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[esNN2Sig Config]
configStr = Generic
formula = "1.+exp(@1*@0)" esNN2 c
c = -3.294 L(-10 - -1)
```

2.2.20 rarGounarisSakurai Configs

`rarGounarisSakurai` is a wrapper of `RooGounarisSakurai` to build a Gounaris-Sakurai pi-pi scattering PDF.

- `configStr = GounarisSakurai ["<pdf Title>"]`
  This config specifies the pdf type is `GounarisSakurai`. This config is required to have this pdf configured as `rarGounarisSakurai`.
- `x = AbsReal Def`
- `mean = AbsReal Def`
- `width = AbsReal Def`
- `spin = AbsReal Def`
- `radius = AbsReal Def`
- `mass_a = AbsReal Def`
- `mass_b = AbsReal Def`
  `x` is the default observable of the pdf. `mean` is the mean of the pdf. `width` is the width of the pdf. `spin` is the spin of the pdf (=0,1,2). Default is 1. After being defined, the valus is always held constant in the fit. `radius` is the form factor radius of the pdf. Default is 3.1/GeV. `mass_a` is the mass of first daughter of the decay. Default is the pi mass. After being defined, the valus is always held constant in the fit. `mass_b` is the mean of second daughter of the decay. Default is the pi mass. After being defined, the values are always held constant in the fit. The default units is GeV; if a different uint is used all the parameters must be suitably scaled. All the `AbsReal` variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[massSig Config]
configStr = GounarisSakurai "#rho K*"
x = mass                    // GeV
mean  = 0.790  L(0.7 - 0.9) // GeV
width = 0.15   L(0.0 - 1.0) // GeV
```

2.2.21 rarHistPdf Configs

`rarHistPdf` is a wrapper of `RooHistPdf` to build PDF using multidimensional histogram.

- `configStr = HistPdf ["<pdf Title>"]`
  This config specifies the pdf type is `HistPdf`. This config is required to have this pdf configured as `rarHistPdf`.
- `obs = <obs1> <obs2> ...`
  Config `obs` lists all observables for this PDF, `<obs1>`, `<obs2>`, ... All the observables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[Sig Config]
configStr = HistPdf
obs = mes de
fitData = sigMC
```

2.2.22 rarKeys Configs

`rarKeys` is a wrapper of `RooKeysPdf/Roo2DKeysPdf` to build 1/2D Keys PDF.

- `configStr = Keys ["<pdf Title>"]`
- `configStr = 2DKeys ["<pdf Title>"]`
  This config specifies the pdf type is `Keys` or `2DKeys`. This config is required to have this pdf configured as `rarKeys`.
- `x = AbsReal Def`
- `y = AbsReal Def`
- `rho = Double_t`
- `keysOption = Options`
  `x` and `y` (for 2D only) are the default observables of the pdf. `rho` is width scale factor. `keysOption` is 1/2D Keys options. For 1D, the options can be any enum value of `RooKeysPdf::Mirror` (NoMirror, MirrorLeft, MirrorRight, MirrorBoth); for 2D, the options can be any valid characters of `Roo2DKeysPdf::setOptions` (a=adaptive (default),n=normal,m=mirror,d=debug,v=verbose,vv=very verbose).

An example is shown below:

```
[emcNNSigSimBoth Config]
configStr = Keys
x         = emcNN2
keysOption = MirrorBoth
```

2.2.23 rarLass Configs

`rarLass` is a wrapper to build the LASS parameterisation of the high mass K*(1430). The default values are taken from Nucl Phys B296, 493 (1988).

- `configStr = Lass ["<pdf Title>"]`
  This config specifies the pdf type is `Lass`. This config is required to have this pdf configured as `rarLass`.
- `x = AbsReal Def`
- `mean = AbsReal Def`
- `width = AbsReal Def`
- `effRange = AbsReal Def`
- `scatlen = AbsReal Def`
- `turnOffVal = AbsReal Def` `x` is the default observable of the pdf. `mean` is the mass of the K*(1430) resonance; `width` is its width; `effRange` is the effective range parameter (default=3.32 GeV/c^2); `scatlen` is the scattering length (default=2.07 GeV/c^2); and `turnOffVal` is the upper mass limit used by LAss in their fit (default=1.65 GeV/c^2);

  All the variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[massSig Config]
configStr=Lass
x           = mass
```

```
mean        = 1.4      L (1.3 - 1.5)
turnOffVal  = 1.53   C L (0.2 - 2.0)
width       = 0.2579 C L (0.2 - 0.5)
effRange    = 0.1119 C L (0.1 - 4.0)
scatlen     = 0.05   C L (0.0 - 4.0)
```

2.2.24 rarPoly Configs

`rarPoly` is a wrapper of `RooPolynomial/RooChebychev` to build Polynomial/Chebychev PDF.

- `configStr = Polynomial ["<pdf Title>"]`
- `configStr = Chebychev ["<pdf Title>"]`
  This config specifies the pdf type is `Polynomial` or `Chebychev`. This config is required to have this pdf configured as `rarPoly`.
- `x = AbsReal Def`
- `nOrder = <orderVal>`
- `P01 = AbsReal Def`
- `...`
- `P<orderVal> = AbsReal Def`
  `x` is the default observable of the pdf. `nOrder` is the order of the polynomial. `P01`, `...`, `P<orderVal>` define all the coeffs. `P00` is not needed. All the `AbsReal` variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[emcNNSigSimEmc Config]
configStr = Polynomial "4th order poly"
x = emcNN2
nOrder = 4
P01 = 1 +/- 10 L(-100 - 100)
P02 = 1 +/- 10 L(-1000 - 1000)
P03 = 0 +/- 10 L(-1000 - 1000)
P04 = 0 +/- 10 L(-1000 - 1000)

[mOmegaPolyBkg Config]
configStr = Chebychev "3rd order Cheby"
x = mOmega
nOrder = 3
P01 = 0.2400 +/- 0.28626 L(-1000 - 1000)
P02 = 0.1 +/- 0.005 L(-100 - 100)
P03 = 0.1 +/- 0.005 L(-100 - 100)
```

2.2.25 rarRelBreitWigner Configs

`rarRelBreitWigner` is a wrapper of `RooRelBreitWigner` to build a relativistic Breit Wigner with Blatt-Weisskopf form factors. References: PDG..

- `configStr = RelBreitWigner ["<pdf Title>"]`
  This configuration specifies the PDF type is `RelBreitWigner`. This configuration is required to have this PDF configured as `rarRelBreitWigner`.
- `x = AbsReal Def`
- `mean = AbsReal Def`
- `width = AbsReal Def`
- `radius = AbsReal Def`
- `mass_a = AbsReal Def`
- `mass_b = AbsReal Def`
- `spin = AbsReal Def`

`x` is the default observable of the PDF. `mean` is the mean of the PDF. `width` is the width of the PDF. `radius` is the meson radius (default = 3.1 GeV^-1). `spin` is the spin of the PDF (= 0,1,2). Default is 1. `mass_a` and `mass_b` are the masses of the daughters of the resonance (e.g. kaon and pion from a K*0). The default for mass_a is 0.4937 GeV (kaon) and for mass_b 0.1396 GeV (pion) After being defined, the spin, radius, mass_a and mass_b are always held constant in the fit. The default assumes that all units are in GeV; if another unit is used then the radius, mass_a and mass_b must all be given. All the `AbsReal` variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[massSig Config]
// in default unts of GeV
configStr = RelBreitWigner "f0(980) K*"
x = mass                      // GeV
mean  = 0.975 L(0.9 - 1.0)    // GeV
width = 0.044 L(0.02 - 1.0)   // GeV
mass_a = 0.1396 C // pi mass
mass_b = 0.1396 C // pi mass
spin = 1

[massSig Config]
// in default units of MeV
configStr = RelBreitWigner "f0(980) K*"
x = mass                      // MeV
mean   = 975 L(900 - 1000)    // MeV
width  = 4.4 L(2 - 1000)   // MeV
mass_a = 139.6 C // pi mass MeV
mass_b = 139.6 C // pi mass MeV
radius = 3100 C // meson radius in MeV
spin  = 1
```

2.2.26 rarStep Configs

`rarStep` is a wrapper of `RooParametricStepFunction` to build Parametric Step Function PDF.

- `configStr = Step ["<pdf Title>"]`
  This config specifies the pdf type is `Step`. This config is required to have this pdf configured as `rarStep`.
- `x = AbsReal Def`
- `nBins = <binVal>`
- `limits = <val1> <val2> ... <val<binVal+1>>`
- `H00 = AbsReal Def`
- `...`
- `H<binVal-1> = AbsReal Def`
  `x` is the default observable of the pdf. `nBins` is the number of bins of the step function. `limits` is a set of `<binVal>+1` Double_t's setting the bounds of those bins. `H00`, `...`, `H<binVal-1>` are `<binVal>-1` free parameters of the step function. All the `AbsReal` variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

  Please make sure the sum of initial values of `Hxx`'s is less than 1, otherwise the PDF may not converge.

An example is shown below:

```
[esNN2Sig Config]
configStr=Step
x=esNN2
nBins=16
limits=0 .1 .2 .3 .4 .5 .6 .65 .7 .75 .8 .85 .9 .93 .95 .975 1.
H00=.05 L(0 - 1)
H01=.05 L(0 - 1)
H02=.05 L(0 - 1)
H03=.05 L(0 - 1)
H04=.05 L(0 - 1)
H05=.05 L(0 - 1)
H06=.05 L(0 - 1)
H07=.05 L(0 - 1)
H08=.05 L(0 - 1)
H09=.05 L(0 - 1)
H10=.05 L(0 - 1)
H11=.05 L(0 - 1)
H12=.05 L(0 - 1)
H13=.05 L(0 - 1)
H14=.05 L(0 - 1)
```

2.2.27 rarTriGauss Configs

`rarTriGauss` is a wrapper to build Triple Gaussian / Triple Gaussian Model / Gexp Shape. It is for convenience of those commonly used composite pdfs related to triple-Gaussian.

For TripleGaussian:

- `configStr = TriGauss ["<pdf Title>"]`
  This config specifies the pdf type is `TriGauss`.
- `x = AbsReal/RealVar Def`
- `meanC = AbsReal Def`
- `sigmaC = AbsReal Def`
- `meanT = AbsReal Def`
- `sigmaT = AbsReal Def`
- `meanO = AbsReal Def`
- `sigmaO = AbsReal Def`
- `fracC = AbsReal Def`
- `fracO = AbsReal Def`
- `msSF = AbsReal Def`
  `x` is the default observable of the PDF. No derived dependent is allowed if you specify `msSF`, in which case, please make sure `x` is observable defined in datasets. If specified, `msSF` is the scale for `meanC`, `sigmaC`, `meanT` and `sigmaT`. All the `AbsReal` parameters can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.
- `FlatScaleFactorIntegral = <yes|no>`
  This optional config specifies to use FlatScaleFactorIntegral or not when `msSF` is specified. The default is `yes`. To disable FlatScaleFactorIntegral, this config must be set to `no`.

For TripleGaussian Model:

- `configStr = TriGaussModel ["<pdf Title>"]`
  This config specifies the pdf type is `TriGaussModel`.
- `x = RooRealVar Def`
- `meanC = AbsReal Def`
- `sigmaC = AbsReal Def`
- `meanT = AbsReal Def`
- `sigmaT = AbsReal Def`
- `meanO = AbsReal Def`
- `sigmaO = AbsReal Def`
- `fracC = AbsReal Def`
- `fracO = AbsReal Def`
- `msSF = AbsReal Def`
  `x` is the default observable of the PDF. No derived dependent is allowed for this PDF so please make sure `x` is observable defined in datasets. `msSF` is the scale for `meanC`, `sigmaC`, `meanT` and `sigmaT`; All the `AbsReal` parameters can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

- `FlatScaleFactorIntegral = <yes|no>`
  This optional config specifies to use FlatScaleFactorIntegral or not. The default is `yes`. To disable FlatScaleFactorIntegral, this config must be set to `no`.

An example is shown below:

For GexpShape:

- `configStr = GexpShape ["<pdf Title>"]`
  This config specifies the pdf type is `GexpShape`.
- `x = RooRealVar Def`
- `tau = AbsReal Def`
- `decayType = <typeName>`
- `meanC = AbsReal Def`
- `sigmaC = AbsReal Def`
- `meanT = AbsReal Def`
- `sigmaT = AbsReal Def`
- `meanO = AbsReal Def`
- `sigmaO = AbsReal Def`
- `fracC = AbsReal Def`
- `fracO = AbsReal Def`
- `msSF = AbsReal Def`
  `x` is the default observable of the PDF. No derived dependent is allowed for this PDF so please make sure `x` is observable defined in datasets. `tau` is the average *B0* lifetime. `decayType` can be `SingleSided`, `DoubleSided` (default), `Flipped`. `msSF` is the scale for `meanC`, `sigmaC`, `meanT` and `sigmaT`; All the `AbsReal` parameters can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.
- `FlatScaleFactorIntegral = <yes|no>`
  This optional config specifies to use FlatScaleFactorIntegral or not. The default is `yes`. To disable FlatScaleFactorIntegral, this config must be set to `no`.

An example is shown below:

2.2.28 rarTwoGauss Configs

`rarTwoGauss` is a wrapper to build Double Gaussian PDF.

- `configStr = TwoGauss ["<pdf Title>"]`
  This config specifies the pdf type is `TwoGauss`. This config is required to have this pdf configured as `rarTwoGauss`.
- `x = AbsReal Def`
- `meanC = AbsReal Def`
- `sigmaC = AbsReal Def`
- `meanT = AbsReal Def`
- `sigmaT = AbsReal Def`
- `fracC = AbsReal Def`

- scale = AbsReal Def
- shift = AbsReal Def
  x is the default observable of the pdf. sigmaC will be scaled by the value of scale if specified; meanC and meanT will be shifted by the value of shift if specified. All the variables can be defined as RooRealVar or RooFormulaVar. See createVar (#item_createVar) for more info on how to create those variables.

An example is shown below:

```
[deSig Config]
configStr = TwoGauss "Two Gaussians"
x  =de
meanC  = 0    L(-.1 - .1)
meanT  = 0    L(-.1 - .1)
sigmaC = 0.02 L(0 - .15)
sigmaT = 0.1  L(0 - .3)
fracC  = 0.8  L(0 - 1)
```

2.2.29 rarUniform Configs

rarUniform is a wrapper of RooUniform to build a flat PDF in N dimensions

- configStr = Uniform ["<pdf Title>"]
  This config specifies the pdf type is Uniform. This config is required to have this pdf configured as rarUniform.
- obs = <obs1> <obs2> ...
  Config obs lists all observables for this PDF, <obs1>, <obs2>, ... All the observables can be defined as RooRealVar or RooFormulaVar. See createVar (#item_createVar) for more info on how to create those variables.

An example is shown below:

```
[Sig Config]
configStr = Uniform
obs       = mes de
fitData   = sigMC
```

2.2.30 rarVoigtian Configs

rarVoigtian implements a Voigtian function. This distribution is a Breit Wigner convoluted with a Gaussian.

- configStr = Voigtian ["<pdf Title>"]
  This config specifies the pdf type is Voigtian. This config is required to have this pdf configured as rarVoigtian.
- x = AbsReal Def
- mean = AbsReal Def
- width = AbsReal Def
- sigma = AbsReal Def

x is the default observable of the pdf. mean is the mean of the Breit-Wigner. sigma is the width of the Breit-Wigner. width is the width of the Gaussian convolution function.

All the variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[voigtianSig Config]
configStr = Voigtian
x     = de
mean  = 0 L (-1 - 1)
sigma = 0.003 L(0 - 1)
width = 0.004 L(0 - 1)
```

2.2.31 rarUsrPdf

`rarUsrPdf` is a wrapper to build user-defined PDF. So for it to work for user-defined PDF, one needs to modify the source code, basically file `rarUsrPdf.cc` in working release dir. Please follow instructions at the very beginning of the source file:

```
// Please change this cc file to make use of your PDF.
// In many cases, you just need to change codes inside marks:
//=================================================================>
//                                                                 v
//                                                                 v
//  and
//                                                                 ^
//                                                                 ^
//=================================================================>
```

There are two main parts need to be changed:

1. Include header file of user-defined PDF

```
//=================================================================>
// Please include your RooFit Pdf header here                      v
// #include "mydir/myPdf.hh"                                       v
//                                                                 ^
//                                                                 ^
//=================================================================>
```

2. Create `RooAbsReal`'s and `RooAbsPdf` in function `rarUsrPdf::init()`

```
//=================================================================>
// change the lines in between the marks as you want               v
//                                                                 v
// first get its obs
_x=createAbsReal("x", "observable"); assert(_x);
// Config pdf params
// instead of a b c etc, you can give them more meaningful
// names and titles for example
// _a=createAbsReal("mean", "#mu", 0, -10, 10);
// _b=createAbsReal("sigma", "#sigma", 0, -10, 10);
// if you give them different names, please use those names
// in the PDF config sections,
// for example, a is now mean, b sigma, etc.
```

```
// [myPdf Config]
// configStr = UsrPdf
// x = AbsReal Def
// mean = AbsReal Def
// sigma = AbsReal Def

// Default param creation
_a=createAbsReal("a", "a", 0, -10, 10);
_b=createAbsReal("b", "b", 0, -10, 10);
_c=createAbsReal("c", "c", 0, -10, 10);
_d=createAbsReal("d", "d", 0, -10, 10);
_e=createAbsReal("e", "e", 0, -10, 10);
_params.Print("v");

// YOU MUST CREATE YOUR PDF AND SET IT TO _thePdf
// create pdf
//_thePdf=new myPdf(Form("the_%s", GetName()),_pdfType+" "+GetTitle(),
// *_x, *_a, *_b, *_c, *_d, *_e);
//                                                                    ^
// change the lines in between the marks as you want                  ^
//===================================================================>
```

Config Directives:

- `configStr = UsrPdf ["<pdf Title>"]`
  This config specifies the pdf type is `UsrPdf`. This config is required to have this pdf configured as `rarUsrPdf`.
- `x = AbsReal Def`
- `a = AbsReal Def` or `<name_chosen_for_a> = AbsReal Def`
- `b = AbsReal Def` or `<name_chosen_for_b> = AbsReal Def`
- `c = AbsReal Def` or `<name_chosen_for_c> = AbsReal Def`
- `d = AbsReal Def` or `<name_chosen_for_d> = AbsReal Def`
- `e = AbsReal Def` or `<name_chosen_for_e> = AbsReal Def`

`x` is the default observable of the pdf. `a`, `b`, `c`, `d`, and `e` are the parameters defined in the `rarUsrPdf::init()`. All the variables can be defined as `RooRealVar` or `RooFormulaVar`. See createVar (`#item_createVar`) for more info on how to create those variables.

An example is shown below:

```
[myPdf Config]
configStr = UsrPdf
x = myObs
a = 36 L(0 - 100)
b = 36 L(0 - 100)
c = 36 L(0 - 100)
d = 36 L(0 - 100)
e = 36 L(0 - 100)
```

2.3 Fitter Actions Configuration

After every pdf is created, fitter action section is used to direct the fitter to finish its jobs. The default action section, `[Fitter Action]`, can be changed by command line option, `-A "<fitter action section>"`. The default for all pdf actions are no action. It is advisable to have several sections with one action per section, and then run the fitter with `-A` option to specify which action to fulfill.

Each action works like a pipe: it has input from other action or initial values, it then outputs param/root files as other actions' input or final results for user to review. Each action initializes its params by reading in intermediate param file, and each action also has configs to override those intermediate params after reading them in.

The param value overriding order is: Initial values from pdf config sections will be overridden by input param file (except for pdfFit action if `prePdfReadParams` and `prePdfReadSecParams` are default); then values from input param file are overridden by values specified in action section and param section with config `xxxReadSecParams`, finally, the constant attributions will be overridden/specified with configs `postPdfFloat`, `preMLFix`, and `preMLFloat` in pdf config and/or action sections. Before params are saved into param files, they will be set to constant.

2.3.1 pdfFit Action

pdfFit action is to fit pdfs to particular datasets to get their parameters. Pdfs, datasets, etc., can be plotted after fitting.

The default param input is the initial values after all PDFs are created. Configs `prePdfReadParams` and `prePdfReadSecParams` can be used to change the initial values. The output param file is the default input for toy study and ML fit. One can override any fitted values from this action by some configs listed below. ROOT file containing plots produced by this action is for user to review the fitting. As always, log file also provides useful information.

- `pdfFit = <no|yes>`
  To enable this action, this config must be explicitly set to `yes` (default `no`).
- `pdfToFit = <pdfName1> [<pdfName2> ...]`
  To fit given pdfs only, list them in this config, and all other pdfs will be ignored. (default list is null, fit all PDFs). This config is useful when you are fine-tuning parameters of one or two pdfs and do not want to run fitting on every pdf.
- `prePdfReadParams = <no|yes|paramFileID>`
  To override initial param values after PDFs are created for pdfFit action, this config must be explicitly set to `yes` (default `no`). The param file name will be automatically generated based on this config. You can specify `paramFileID` (`#item_paramFileID`) to change its default name.
- `prePdfReadSecParams = <no|yes|secName>`
  If set to `yes` (default `no`), just before pdfFit action, the fitter will try to override the params with values found in the action section. Instead of `yes` or `no`, you can also give it a section name, and then the fitter will try to read values from that section first. This config is useful when you want to override any initial values before pdfFit.
- `postPdfMakePlot = <no|yes|rootFileBaseName>`
  To get pdf plots after pdfFit, this config must be explicitly set to `yes` (default `no`). The

root file name will be `<configFile>.<fitterName>.pdfPlot.<actionName>.root` unless the token is actually a root file name.

- `postPdfReadSecParams = <no|yes|secName>`
  If set to `yes` (default `no`), just after pdfFit is done, the fitter will try to override the params with values found in the action section. Instead of `yes` or `no`, you can also give it a section name, and then the fitter will try to read values from that section first. This config is useful when you want to override any values got by the fit with those you think are more appropriate, for example, those got from control samples.
- `postPdfWriteParams = <no|yes|paramFileID>`
  To output pdf parameters after pdfFit, this config must be explicitly set to `yes` (default `no`). The param file name will be automatically generated based on this config. You can specify `paramFileID` (`#item_paramFileID`) to change its default name.
- `fitRange_<obsName> = <Min> <Max>`
  It sets **pdfFit** ranges for observable `<obsName>`. If not specified, the range will be set to the full ranges when it is created. This config will affect all `pdfFit`, if it is needed for just one or two PDFs, please move this config to individual PDF config sections.
- `plotRange_<obsName> = <Min> <Max>`
  It sets plot ranges for observable `<obsName>`. If not specified, the range will be set to the full ranges when it is created. This config will affect all `pdfFit`, if it is needed for just one or two PDFs, please move this config to individual PDF config sections.

`prePdfReadParams` has been dropped. The reason for this is that pdfFit is really the very first step in the fitting procedure and you really want to make sure everything you put here is what you mean. So place the initial values into the config file and make sure they are reasonable. With another file overriding the config file, I can only see more trouble and subtlety. A corollary is that never change the intermediate param files, and if you want to change the param values in those files, change them in the config file.

An example is shown below:

```
[pdfFit Config]
// pdfFit options
pdfFit = yes
//pdfToFit = deSig deBkg
postPdfMakePlot    = yes
postPdfWriteParams = yes
```

2.3.2 toyStudy Action

toyStudy action is used to validate the fitting procedure. To split a large toy job into several smaller ones, different command line option `-t <toyID>` must be given for different jobs to enable different random seeds, output root file names, etc.

The default param input is the intermediate param file from pdfFit action. Configs `preToyReadParams` and `preToyReadSecParams` can be used to change the initial values. (After unblind, you may want to change the input to the intermediate param file from mlFit action. See config `preToyReadParams` below for howto.) There is no param file output. ROOT file containing toy study results produced by this action is for user to review. As always, log file also provides useful information.

If all or part of the events of a component are embedded from datasets other than generated from PDF, the pulls of yields are calculated based on wrong true value by `RooFit`, in this case, please use pull calculated by `RooRarFit`, with suffix `_embd` after the pull name. For example, yield `nSig`'s pull will be `nSigpull_embd`.

- `toyStudy = <no|yes>`
  To enable this action, this config must be explicitly set to `yes` (default `no`).
- `preToyReadParams = <yes|no|paramFileID>`
  NOT to read in initial values of parameters before toyStudy, this config must be explicitly set to `no` (default `yes`). The param file name will be automatically generated based on this config. You can specify `paramFileID` (`#item_paramFileID`) to change its default name. The default input param file for toy study is the output param file from pdfFit. To read, for example, from output param file of mlFit, set this config like

  ```
  preToyReadParams = A mlFit
  ```
- `preToyReadSecParams = <yes|no|secName>`
  If set to `yes` (default), just after read in params from param file, the fitter will try to override the params with values found in the action section. Instead of `yes` or `no`, you can also give it a section name, and then the fitter will try to read values from that section first. This config is useful when you want to override any values (for example, yields) just for toy studies.
- `preToyRandParams = no|<param1> <formula1> ...`
  It is helpful if we can choose randomly (ie, scan) values for some params before each toy experiment. For example, we want to study the fit behavior for all possible C and S for CP analysis. Or we want to study all possible polarization fractions for B to VV modes. You can list all those params here and give them formulae to specify values. If the first token is `no`, the procedure is disabled.
- `preToyRandGenerators = <pdf1> ...`
  This config specify Pdfs to generate random params.
- `postToyWriteParams = <yes|no|rootFileBaseName>`
  NOT to write out pdf parameter pulls after toyStudy, this config must be explicitly set to `no` (default `yes`). The root file name will be `<configFile>.<fitterName>.toyPlot.<actionName>.%02d.root` unless the token is actually a root file name.
- `protDatasets = <datasetName> <comp1ProtData> <comp2ProtData> ...`
  This config can be set to names of prototype datasets for toy study. (default datasets will be used if necessary but not specified here). You can specify for each component its own protDataset. Prototype datasets are used to get observable distribution information not in ml model. If the ml model is complete, ie, all distribution information is specified within the model, protDataset will not be used.
- `protDataGenLevel = <1|0|2|3>`
  This config specifies how the protDataset for each experiment is generated (default level `1`).
  - `0` No protDataset. If needed, generated from protData generator.
  - `1` Use default master protDataset as protDataset if needed.
  - `2` Generate protDataset from individual protDatasets if needed.

  - 3 Generate protDataset from protData generator if needed.

- `protDataVars = [<obs1> ...]`
  Please specify this config in the section where corresponding Pdf is created. See Section 2.2.1 [commonPdfConfig], page 17.
- `protDataEVars = [<obs1> ...]`
  Please specify this config in the section where corresponding Pdf is created. See Section 2.2.1 [commonPdfConfig], page 17.
- `toyNexp = <numberOfToy>`
  To specify number of toy experiments for toyStudy (default `1`). If you specify command line option, `-n <toyNexp>`, the value specified at command line will override this config.
- `toyNevt = <numberOfEvt> <fixed|floated|extended|notfixed>`
  To specify the number of events per toy experiment and Poisson fluctuation (default `0 fixed`). `0` means the number of events is that of the prototype dataset. If the total yields (expected number of events for extended model) is different from the prototype dataset, the largest yield (usually continuum background) will be adjusted. To enable Poisson fluctuation, the second argument should be set to any of `floated`, `extended`, and `notfixed`.
- `toySrc_<srcCoeff> = <srcData1> <val1> ...`
  This config specifies generation sources for component of `<srcCoeff>`. If not specified, the default is always `pdf <srcCoeff_value>` which will generate the component using pdf.

  `<srcCoeff>` is the final variable of component coeff. Component coeff is usually the number of events for that component, and if that number is actually the final variable (not dependent on other variables), its name should replace `<srcCoeff>`. If the number of events is actually a function of branching fraction, etc., the name of branching fraction, etc., should be used.

  `<srcData1>` is the data source to generate toy sample from. The default value is `pdf` which means generating from PDF. Or it could be the name of any dataset defined, and the toy events for this component will be selected from that dataset. `<val1>` is the value of `<srcCoeff>` for that data source. It could be any number valid for `toySrc_<srcCoeff>`. For example, if `<srcCoeff>` is the component coeff itself, `<val1>` is number of events; if it is branching fraction, `<val1>` is the value of branching fraction you want to generate. You can have more than one such source and value pair if you want to generate that component from multi sources.

  `<valX>` can also be a string used to calculate the value as for `RooFormulaVar`. For example, it can be "`@0*@1 nSig fracL`", which means the value for that data source will be `nSig*fracL`.
- `toyEmbdUnCorrelate = <obsGroup1>...`
  This config, if specified, uncorrelate observables among groups for embedded events. For example config

  ```
  toyEmbdUnCorrelate = de mes "fisher mOm"
  ```

  uncorrelate de, mes, "fisher mOm" among each other, but keep the correlation of fisher and mOm. If there are any observables left, the correlation among those observables are also kept.

- `postEmbdRandObs = <srcData1> <obs1> <obs2>...<pdfForSrcData1> ...`
  This config specifies for each srcData (`<srcData1>`, etc., non-pdf source) observables (`<obs1>`, `<obs2>`, etc.) need to be randomized according to pdf (`<pdfForSrcData1>`). This feature is useful to create embedded events for different `C` and `S` from non CP violating MC samples.
- `toyDataFilePrefix = <default|no|yes|toySampleNameID>`
  The toy sample file name will be automatically generated based on this config (See `toySampleNameID` (`#item_paramFileID`)). Since the toy data files are seldom used elsewhere and consume very large disk space, the default (not specified or set to `default`) will be no toy data file output.

  Use command line option `-d <toySampleDir>` to specify a dir with large disk space for toy samples (default dir `.toyData`).
- `toyGenerate = <yes|no>`
  NOT to generate toy sample in toyStudy, this config must be explicitly set to `no` (default `yes`).

  The first toy sample generated will be saved as "`toySample`", so in the same action section, other actions following `toyStudy` can use the sample, for example, to do sPlot. Just give `toySample` as config for input data, for example

  ```
  sPlotData = toySample
  ```
- `toyFit = <yes|no>`
  NOT to do fit in toyStudy, this config must be explicitly set to `no` (default `yes`).
- `toyFitOption = <fitOptions>`
  This config can be used to set ml-fitting options for toy study (default `"emhqr"`). The full ml model is an extended pdf, so the fitting option should always have option `"e"`, which means extended fit.
- `toyFitMinos = <notSet|minosArgs>`
  This config can be used to set MINOS on for only certain parameters for toy study, (default `"notSet"`).
- `toyChkNegativePdf = <no|yes>`
  If set to `yes`, the fitter will check if the PDFs have negative values over the allowed observable ranges. The default value is `no`, not to do the check. If the PDFs have negative value, toy studies will usually show biases in purely PDF generated samples, so this check is not necessary to demonstrate the problem. But it can be used as auxiliary method to confirm the bias (from pure toys) is from negative PDF values. However, it is still possible that it fails to report such problems if subtle PDFs are used.

An example is shown below:

```
[toyStudy AConfig]
// toy options
toyStudy = yes
// init values for yields
nSig   = 160 L(0 - 300)
nChmls = 1703 L(0 - 30000)
nBkg   = 5000 L(0 - 30000) // largest number will be adjusted
preToyRandParams = dtSig_C "@0*cos(@1) R theta" dtSig_S "@0*sin(@1) R theta"
```

```
preToyRandGenerators = Rpdf thetaPdf
protDatasets         = onData //names of protDatasets
toyNexp              = 400 // # experiments
toyNevt              = 0 fixed // default: 0      set to protData #evt
                               //          fixed  no fluctuation
// if it is pure toy, you do not need to specify how to generate
// for embedded toy, for example you can do
toySrc_nSig   = sigMC "@0 nSig" // from sigMC
toySrc_nChmls = bbMC 10 pdf "@0-10 nChmls" // 10 from bbMC, the rest from pdf

[Rpdf Config]
configStr = Generic
formula = "@0*@0" R
R = R 0 L(0 - 1)
[thetaPdf Config]
configStr = Generic
formula = "1" theta
theta = theta 0 L(-3.14159265358979323846 - 3.14159265358979323846)
```

2.3.3 mlFit Action

mlFit action is to fit full ml model to (on-peak) dataset to get yields, etc.

The default param input is the intermediate param file from pdfFit action. Configs `preMLReadParams` and `preMLReadSecParams` can be used to change the initial values. The intermediate param output file have the final values from the ML fit, and will be used as input for all kind of plotting. There is no ROOT file output. All the results are shown in log file.

- `mlFit = <no|yes>`
  To enable this action, this config must be explicitly set to `yes` (default `no`).
- `mlFitData = <datasetName> ["<optional cut string>"]`
  This config specifies the dataset for mlFit action. You can give this config an optional string as the second token, which will be applied to the dataset for additional cuts.
- `mlFitOption = <fitOptions>`
  This config can be used to set ml-fitting options (default `"ehr"`). The full ml model is an extended pdf, so the fitting option should always have option `"e"`, which means extended fit. Add option `"b"` to suppress RooFit informational messages for blind fit.
- `preMLReadParams = <yes|no|paramFileID>`
  NOT to read in initial values of parameters before mlFit, this config must be explicitly set to `no` (default `yes`). The param file name will be automatically generated based on this config. You can specify `paramFileID` (`#item_paramFileID`) to change its default name.
- `preMLReadSecParams = <yes|no|secName>`
  If set to `yes` (default), just after read in params from param file, the fitter will try to override the params with values found in the action section. Instead of `yes` or `no`, you can also give it a section name, and then the fitter will try to read values from that section first.

- `postMLWriteParams = <yes|no|paramFileID>`
  NOT to output pdf paramters after mlFit, this config must be explicitly set to `no` (default `yes`). The param file name will be automatically generated based on this config. You can specify `paramFileID` (`#item_paramFileID`) to change its default name.
- `postMLSignf = [no] <paramName1> <0|0signfValue> ...`
  If specified (and the first token is not `no`), significance of given params with respect to given values (default `0`) will be calculated.
- `postMLSysParams = <no|[<defaultVariantUnit>] param1Specification...>`
  This config specifies how to vary fixed parameters to study the systematic uncertainties of floating parameters we are interested in, due to the uncertainties of those fixed parameters. Optional `<defaultVariantUnit>` specifies default variations in term of errors (default `1`). `paramXSpecification` includes parameter name, plus variation and minus variation in term of parameter errors. If the variations are in term of absolute value, append those numbers with `V`. If only one variation for the param is specified, it will be used for plus and minus variations. If no variation value specified for a param, the default value will be used. For split params, there are two choices. One is to list the unsplit params and vary the split params all together in the same direction. One is to list individual split params and vary them separately. If this config is not specified (or the first token is `no`), no such systematic error study will be performed.
- `postMLSysVars = <no|var1...>`
  This config specifies free parameters, usually those we are interested in, for example, yields, the fitting uncertainties of which, due to uncertainties of fixed parameters in `postMLSysParams`, need to be studied. If not specified (or the first token is `no`), no such systematic error study will be performed.

  When both `postMLSysParams` and `postMLSysVars` are set properly, systematic error studies for `postMLSysVars` due to the variations of fixed `postMLSysParams` will be performed. The systematic error table will be printed out at the end of `mlFit` job. The systematic error table looks like:

```
Systematic Error Table:
                                nSig  corr matrix:   var1  var2  var3
  var1  +4  -2   +.2  -.2    +.2                     1     0     0
  var2  +1  -1   -.3  +.3    -.3                     0     1     0
  var3  +1  -1   +.4  -.4    +.4                     0     0     1
(w/o) corr):                  .5
(w/ corr):                    .5
```

  The first column is the names of fixed vars to vary. The second column is the plus variation for the vars. The third column is the minus variation for the vars. The next three columns are for the first studied var: the plus variation, minus, and average effects. The remaining studied vars, if any, will follow it in the same fashion. At the far right side of the table is the correlation matrix of those fixed vars. The last two rows show the total effects without and with taking the correlation matrix into account.

  Please notice that if the number of fixed vars in the study is large, the table will be very very long and wide, please make sure the editor used to view the table does not wrap lines so that it is more readable.
- `postMLGOFChisq = <no|yes>`

If specified (and the first token is not `no`), Goodness-of-fit defined as chisq will be calculated.

This config can also be used in `toyStudy` action.

An example is shown below:

```
[MLFit Config]
// mlFit options
mlFit = yes
//mlFitOption = emhr
postMLSignf     = nSig
postMLSysParams = deSig_scale .05V deSig_simga
postMLSysVars   = nSig fL
```

2.3.4 scanPlot Action

scanPlot action is to scan NLL within interested param space. The recorded values in dataset or plot for scanPlot are actually 2NLL so they are chisquares.

The default param input is the intermediate param file from mlFit action. `preScanPlotReadParams` and `preScanPlotReadSecParams` can be used to change the initial values. There is no intermediate param file output. ROOT file containing scanPlots produced by this action is for user to review. As always, log file also provides useful information.

RooRarFit provides a set of static functions to manipulate (combine/shift/add errors) the NLL curves (for 1D scanPlot). combine.cc (`Sample_scripts/combine.cc`) and combine.C (`Sample_scripts/combine.C`) show the usage. (Copy `combine.C` to workdir and edit it to use.)

- `scanPlot = <no|yes>`
  To enable this action, this config must be explicitly set to `yes` (default `no`).
- `preScanPlotReadParams = <yes|no|paramFileID>`
  NOT to read in initial values of parameters before scanPlot, this config must be explicitly set to `no` (default `yes`). The param file name will be automatically generated based on this config. You can specify `paramFileID` (`#item_paramFileID`) to change its default name.
- `preScanPlotReadSecParams = <no|yes|secName>`
  If set to `yes` (default `no`), just after read in params from param file, the fitter will try to override the params with values found in the action section. Instead of `yes` or `no`, you can also give it a section name, and then the fitter will try to read values from that section first.
- `scanPlotFile = <default|scanPlotOutputFile.root>`
  The root file name will be `<configFile>.<fitterName>.scanPlot.<actionName>.root` unless the token is actually a root file name.
- `scanPlotData = <datasetName>`
  This config specifies the dataset for scanPlot action.
- `scanPlotFitOption = <fitOptions>`
  This config can be used to set scanPlot fit options (default `"emhr"`). The full ml model

is an extended pdf, so the fitting option should always have option "e", which means extended fit.

- scanVarShiftToNorm = <no|yes>
  The scanPlot min shifts a little bit from the normal mlFit values, to correct the shifts, this config must be set to yes (default no).
- scanVars = <varName1> [xmin] [xmax] ...
  To specify scanPlot variables, this config must be explicitly set to the variable names (default notSet). The variable name should be full name, and for each variable, two optional limits can be specified to get plot ranges rather than the default ones.
- nScanPoints = <100|#Points>
  To specify number of scan points (default 100). If there are more than one var specified in scanVars to scan, the scan points will be drawn randomly in the n-dimensional space. If there is only one var to scan, the points are equally distributed in the allowed range.
- nScanSegments = <1|#times>
  This config is to specify number of scan segments (for 1D only and default is 1). It is useful if the scanPlot action takes too much time if it is done in one job. One can then divide the whole region with this config into smaller pieces and use submitToy to run the jobs with batch queues.

  nScanPoints is the number of scan points per segment so the total scanned points will be nScanPoints*nScanSegments.

An example is shown below:

```
[BrScan]
// scanPlot options
scanPlot = yes
scanVars = BR 0 10
nScanPoints = 50
```

2.3.5 projPlot Action

projPlot action is to get projection plots.

The default param input is the intermediate param file from mlFit action. preProjPlotReadParams and preProjPlotReadSecParams can be used to change the initial values. There is no intermediate param file output. ROOT file containing projPlots and LLR plots produced by this action is for user to review. As always, log file also provides useful information.

- projPlot = <no|yes>
  To enable this action, this config must be explicitly set to yes (default no).
- preProjPlotReadParams = <yes|no|paramFileID>
  NOT to read in initial values of parameters before projPlot, this config must be explicitly set to no (default yes). The param file name will be automatically generated based on this config. You can specify paramFileID (#item_paramFileID) to change its default name.
- preProjPlotReadSecParams = <no|yes|secName>
  If set to yes (default no), just after read in params from param file, the fitter will try to override the params with values found in the action section. Instead of yes or no,

you can also give it a section name, and then the fitter will try to read values from that section first.

- `projPlotFile = <default|projPlotOutputFile.root>`
  The root file name will be `<configFile>.<fitterName>.projPlot.<actionName>.root` unless the token is actually a root file name.
- `projComps = <compName1> ...`
  To specify components to be projected, this config must be explicitly set to the names of those components. You can have more than one component to be projected, just list them here.
- `projLLRPlots = <yes|no|compName1 ...>`
  If set to `yes` (by default), likelihood ratio plots will be done for default dataset, total model, and each component. If you do not want LLR plots of all components, you can specify the components you want here.
- `projLLRScale_<compName> = <scale>`
- `projLLRScale = <scale>`
  It specifies scale factor for `<compName>` in the LLR plot to its expected number of events (default `10`, ie, the normalization factor is 10 times the expected events). The first form has higher priority for a given component.
- `plotBins_LLR = <nBin>`
  It specifies number of plot bins for likelihood ratio plots (default `100`).
- `projVars = <varName1> ...`
  To set the projection observables, this config must be explicitly set to the names of that observables (default `notSet`).
- `projPlotData_<obs> = <datasetName> ["<optional cut string>"]`
- `projPlotData = <datasetName> ["<optional cut string>"]`
  This config specifies the dataset for projection plot action. You can give this config an optional string as the second token, which will be applied to the dataset for additional cuts. This optional cut (if any) will be appended to `projOptimRange`. The first form has higher priority for a given observable.
- `projPlotSaveLLR = <no|yes>`
  If this config is set to `yes` (default `no`), the dataset for projection (specified with `projPlotData` or `projPlotData_<obs>`) will be saved into the output root file with LLR values appended. The LLR column will be named as `lRatioFunc_<var>`.
- `projPlotCat_<obs> = <no|CatName1...>`
- `projPlotCat = <no|CatName1...>`
  Projection plot will also be done for each type of the cats specified here (default `no`).
- `projAsymPlot_<obs> = <no|CatName>`
- `projAsymPlot = <no|CatName>`
  To have asym plot with projection, this config must be set to the cat name (default `no`, no asym plot). The category must have, and can only have, two types, in which case you get one projection plot for each cat type, and one asym plot. The first form has higher priority for a given observable.
- `projLRatioCut_<obs> = <cutVal>`

- `projLRatioCut = <cutVal>`
  To specify likelihood ratio cut. The value should be between 0 and 1 (default `0.9`). The first form has higher priority for a given observable.
- `plotBins_<varName> = <nBin>`
  To specify number of plot bins for projection plot of observable `<varName>`. If not specified, the number of plot bins will be set to the default value for that observable (when the observable is created with argument `B(<nBins>)`).
- `plotRange_<obsName> = <Min> [bBoundary1] ... <Max>`
  It sets plot ranges for observable `<obsName>`. If not specified, the range will be set to the full ranges when it is created. You can specify optional boundaries for asym plot.
- `projFindOptimCut_<obs> = <no|yes>`
- `projFindOptimCut = <no|yes>`
  If set to `yes` (default `no`), the fitter will try to find the optimal `projLRatioCut` for projection. The first form has higher priority for a given observable.
- `projOptimStep_<obs> = <stepVal>`
- `projOptimStep = <stepVal>`
  This config specifies the searching step for optimal cut (default 0.005, ie, 200 steps from ratio 0 to 1). The first form has higher priority for a given observable.
- `projOptimFormula_<obs> = [<optimal_formula>]`
- `projOptimFormula = [<optimal_formula>]`
  The default optimization formula is `N^2/N_total`. You can specify a different formula to optimize, for example, `"pow(@0,3)/@1"`, where `@0` is for `N` and `@1` is for `N_total`, and the optimization is done with `N^3/N_total`. The first form has higher priority for a given observable.
- `projOptimRange_<obs> = [<range_cut>]`
- `projOptimRange = [<range_cut>]`
  This config specifies ranges on which the optimization is performed. The default is '1' which means the full range (of projection variable, etc.). The first form has higher priority for a given observable.
- `projOptimData = [<comp1Data> ...]`
  This config specifies the dataset for each component to be projected from where the projLRatioCut efficiency for that component is got. If not specified, the datasets used are the default datasets.
- `projOptimDataLimit = [40000|<NumOfEvts>]`
  This config specifies the max number of events (default `40000`) for datasets used for LLR cut optimization.

An example is shown below:

```
[ProjAct]
// projectionPlot options
projPlot = yes
preProjPlotReadParams = yes
projPlotFile = default
projComps = SigPdf
```

```
projVars = de mes
// for de
projLRatioCut_de = .45
plotBins_de = 20
projFindOptimCut_de = yes
projOptimStep_de = .001
projOptimRange_de = "abs(de)<0.07"
// for mes
projLRatioCut_mes = .55
plotBins_mes = 16
projFindOptimCut_mes = yes
projOptimStep_mes = .001
projOptimRange_mes = "mes>5.274&&mes<5.286"
// for all other implicitly
projLRatioCut = .85
projFindOptimCut = no
projOptimStep = .005
```

2.3.6 contourPlot Action

contourPlot action is to get 2NLL contour plots for two floating parameters.

The default param input is the intermediate param file from mlFit action. `preContourPlotReadParams` and `preContourPlotReadSecParams` can be used to change the initial values. There is no intermediate param file output. ROOT file containing contourPlots produced by this action is for user to review. As always, log file also provides useful information.

- `contourPlot = <no|yes>`
  To enable this action, this config must be explicitly set to `yes` (default `no`).
- `preContourPlotReadParams = <yes|no|paramFileID>`
  NOT to read in initial values of parameters before contourPlot, this config must be explicitly set to `no` (default `yes`). The param file name will be automatically generated based on this config. You can specify `paramFileID` (`#item_paramFileID`) to change its default name.
- `preContourPlotReadSecParams = <no|yes|secName>`
  If set to `yes` (default `no`), just after read in params from param file, the fitter will try to override the params with values found in the action section. Instead of `yes` or `no`, you can also give it a section name, and then the fitter will try to read values from that section first.
- `contourPlotFile = <default|contourPlotOutputFile.root>`
  The root file name will be `<configFile>.<fitterName>.contPlot.<actionName>.root` unless the token is actually a root file name.
- `contourPlotData = <datasetName> ["<optional cut string>"]`
  This config specifies the dataset for contourPlot action. You can give this config an optional string as the second token, which will be applied to the dataset for additional cuts.

- `nContours = <2|1|3|4|5|6>`
  To specify number of contours. The value should be between 1 and 6 (default `2`). Each contour is 1 sigma (unit in terms of 2NLL) away from inner contour.
- `contourVars = <varNameX> [xmin] [xmax] <varNameY> [ymin] [ymax]`
  To specify contour variables, this config must be explicitly set to the variable names (default `notSet`). The variable name should be full name, and for each variable, two optional limits can be specified to get plot ranges rather than the default ones.
- `contourRestrictFloatParams = <no|yes|nErrorForAll|floatParam1 nError1 ...>`
  It might save a lot of time and have a better chance to get a contour plot if we restrict all other free parameters to narrow ranges. If set to `yes` (default `no`), the limits of every float parameter will be set to 2 times its error (or remains unchanged if that limit is larger). You can give a different scale factor for all floating parameters, or you can give explicitly for each floating parameter a range factor. The factor can be any positive number.

An example is shown below:

```
[fLvsNsigContour]
// contourPlot options
contourPlot = yes
preContourPlotReadParams = yes
contourPlotFile = default
contourVars = nSig fL
nContours = 3
```

2.3.7 sPlot Action

sPlot action is to get sPlots.

The default param input is the intermediate param file from mlFit action. `preSPlotReadParams` and `preSPlotReadSecParams` can be used to change the initial values. There is no intermediate param file output. ROOT file containing sPlots and sWeighted datasets produced by this action is for user to review. As always, log file also provides useful information.

- `sPlot = <no|yes>`
  To enable this action, this config must be explicitly set to `yes` (default `no`).
- `preSPlotReadParams = <yes|no|paramFileID>`
  NOT to read in initial values of parameters before sPlot, this config must be explicitly set to `no` (default `yes`). The param file name will be automatically generated based on this config. You can specify `paramFileID` (`#item_paramFileID`) to change its default name.
- `preSPlotReadSecParams = <no|yes|secName>`
  If set to `yes` (default `no`), just after read in params from param file, the fitter will try to override the params with values found in the action section. Instead of `yes` or `no`, you can also give it a section name, and then the fitter will try to read values from that section first.

- `sPlotFile = <default|sPlotOutputFile.root>`
  The root file name will be `<configFile>.<fitterName>.sPlot.<actionName>.root` unless the token is actually a root file name.
- `sPlotSaveSWeight = <no|yes>`
  By default, only the sPlot itself will be saved into root file. If this config is set to `yes` (default `no`), the sWeighted dataset will be saved as well.
- `sPlotComps = <all|compName1 ...>`
  An sPlot will be plotted for each component specified. The default is `all`, which means sPlot will be plotted for all components.
- `sPlotVars = <varName> ...`
  To set the sPlot observables, this config must be explicitly set to the names of that observables (default `notSet`).
- `sPlotData_<obs> = <datasetName> ["<optional cut string>"]`
- `sPlotData = <datasetName> ["<optional cut string>"]`
  This config specifies the dataset for sPlot action. You can give this config an optional string as the second token, which will be applied to the dataset for additional cuts. The first form has higher priority for a given observable.
- `sPlotIgnoredVars_<obs> = <varName1> ...`
- `sPlotIgnoredVars = <varName1> ...`
  In addition to `sPlotVar`, observables listed here (if any) will be ignored for sPlot plotting. The reason is some observables are highly correlated, to remove one, you have to remove all, otherwise the PDF for sPlot will not be correct, for example, the two helicities in VV modes. The default is `notSet` for no additional removed observables. The first form has higher priority for a given observable.
- `sPlotPdfOverlay_<obs> = <direct|yes|no|[yield1 pdf1...]>`
- `sPlotPdfOverlay = <direct|yes|no|[yield1 pdf1...]>`
  The default is to overlay pdf on sPlot. If set to `no`, no pdf overlaid; if set to `direct` (default), the pdf used will be the pdf built directly for that component and that obs, instead of the total pdf; if set to `yes`, the total component pdf will be used, where the time of pdf overlay will be long due to the projection of the total pdf in most cases. You can also give explicitly for each component which pdf to use, or if to plot at all. For example,

  ```
  sPlotPdfOverlay_mRho = direct nSig mRhoSigExtra nBkg no
  ```

  specifies `mRho` sPlot pdf overlay settings: no overlay for yield `nBkg`, use 'direct' pdfs for all other components and use pdf `mRhoSigExtra` as the 'direct' pdf for yield `nSig`.

  To overlay pdf onto sPlot of obs which is *not* included in the fit model, you have to give explicitly the name of pdf to overlay for that yield and obs, because the fitter can not find overlay pdf using the fit model.
- `sPlotNormIgnoredObs = <obs1> ...`
  This config can be used to ignore some observables for pdf normalization. For example, when plot with dt, `dtErr` and `tagFlav` should not be in the normalization list so this config should be used.
- `plotBins_<varName> = <nBin>`
  To specify number of plot bins for sPlot of observable `<varName>`. If not specified,

the number of plot bins will be set to the default value for that observable (when the observable is created with argument `B(<nBins>)`).

- `plotRange_<obsName> = <Min> <Max>`
  It sets plot ranges for observable `<obsName>`. If not specified, the range will be set to the full ranges when it is created.
- `sPlotHist = <default|no|yes>`
  By default (`default` or `no`), the sPlot is saved as RooPlot. If for some reason, it is desirable to save the sPlot as regular histogram, this config can be set to `yes`.

An example is shown below:

```
[desPlot Config]
// sPlot options
sPlot = yes
preSPlotReadParams = yes
sPlotFile   = default
sPlotComps  = all
sPlotVar    = de
plotBins_de = 15
```

2.3.8 combinePlot Action

The combinePlot action is used to combine the NLL curves from the scan action and convoluted them with systematic errors. The systematic errors can be categorized as additive errors, uncorrelated multiplicative errors and correlative multiplicative errors. The significance of the central value including the combined systematic and statistical errors can be calculated as well as an upper limit at a chosen confidence level.

- `combinePlot = <no|yes>`
  To enable this action, this config must be explicitly set to `yes` (default `no`).
- `combineNcurves = <#ncurves>`
  The number of curves to combine.
- `combineFilenames = <filenames>`
  The names of the root files produced by the scan action for each curve. Number of filenames must be the same as the number of curves. The root file name is usually something like `<configFile>.<fitterName>.scanPlot.<actionName>.root`
- `combinePlotnames = <Rooplot name>`
  The names of RooPlots in the scan rootfiles. Number of names must be the same as the number of curves. The names is usually something like "NLLScanPlot_nSig".
- `combineAdditive = <#values>`
  The additive systematic errors for each of the curves. If the errors are symmetric, then one error is required for each curve. If asymmetric errors are required then the negative and positive errors should be given. Symmetric and asymmetric errors can not be given together (if you need to do this then write the symmetric error as an asymmetric error with the same negative and positive error

  An example is shown below:

  ```
  combineNcurves = 2
  combineAdditive  = 0.5 0.46 // symmetric errors or
  ```

```
// symmetric errors written as asymmetric errors:
//combineAdditive = -0.50 0.50 -0.46 0.46
//combineAdditive = -0.52 0.51 -0.48 0.47 // asymmetric errors
```

- `combineMultiplicativeUncorrelated = <#values>`
  The uncorrelated multiplicative systematic errors for each of the curves. If the errors are symmetric, then one error is required for each curve. If asymmetric errors are required then the negative and positive errors should be given. Symmetric and asymmetric errors can not be given together (if you need to do this then write the symmetric error as an asymmetric error with the same negative and positive error

  An example is shown below:

```
combineNcurves = 2
combineMultiplicativeUncorrelated  = 0.5 0.46 // symmetric errors or
// symmetric errors written as asymmetric errors
//combineMultiplicativeUncorrelated = -0.50 0.50 -0.46 0.46
// asymmetric errors
//combineMultiplicativeUncorrelated = -0.52 0.51 -0.48 0.47
```

- `combineMultiplicativeCorrelated = <#values>`
  The uncorrelated multiplicative systematic errors for each of the curves. Only symmetric errors can be given.

  An example is shown below:

```
combineNcurves = 2
combineMultiplicativeCorrelated  = 0.5 0.46 // symmetric errors
```

- `combineFitBias = <#values>`
  If the fit bias has not been applied in the fitting stage it can subtracted from the central value here. The default value is not to apply a correction. Number of values must be the same as the number of curves.
- `combineUpperLimit = <yes|no CL>`
  If the first value is set to "yes" the code will calculate a Upper Limit Confidence Level. The default value is "yes 90" which will calculate a 90% CL upper limit.
- `combineXaxisTitle = <X axis title>`
  You can specify a title for the x-axis.

An example is shown below:

```
[CombineAct]
combinePlot = yes
combineNcurves = 2 // combine two curves
combineFilenames = "results/RhozKp_Kz.mlFitter_Config.scanPlot.rhoBRScan.root" \\
                   "results/RhozKp_Kp.mlFitter_Config.scanPlot.rhoBRScan.root"
combinePlotnames = "NLLScanPlot_measBR_Sig" \\
                   "NLLScanPlot_measBR_Sig"
combineCentralValues = 4.73 3.52 // central value e.g. branching fraction x 10^-6
combineAdditive     =  0.5004 0.460768 // symmetric errors for 2 curves
combineMultiplicativeUncorrelated = 0.00946 0.007 // uncorrelated multiplicative error
combineMultiplicativeCorrelated = 0.211 0.1858 // correlated multiplicative errors
//combineFitBias  = 0.0 0.0 // fit bias if not already corrected for. (default)
```

```
combineUpperLimit = yes 90 // calculate 90% C.L. upper limit (default)
combineXaxisTitle = "Branching Fraction (#times 10^{-6})"
```

2.4 Sample Configurations

You can find sample configuration files here (`http://rarfit.sourceforge.net/Sample_configs`) (Babar internal only). Please copy the sample config file and its dsd file to `workdir` to run. A few typical config files are explained below:

- omks.config (`http://rarfit.sourceforge.net/Sample_configs/omks.config`) Config file for omegaKz analysis. You can find pretty much everything for the analysis, yield fit, BR fit, S, C, toy study (scanning, etc.), etc.
- omh.config (`http://rarfit.sourceforge.net/Sample_configs/omh.config`) Config file for omegaH analysis. You can find configs for K/pi fitting.
- omrho.config (`http://rarfit.sourceforge.net/Sample_configs/omrho.config`) Config file for omegaRho analysis. You can find configs for B–>VV analysis.
- ksboth.config (`http://rarfit.sourceforge.net/Sample_configs/ksboth.config`) Config file for eta'Ks analysis. You can find configs for physCat splitting.

3 RooRarFit Tutorial File

3.1 Introduction

RooRarFit comes with example configuration files that can be used for regression testing and to test out the RooRarFit commands. The files are available in *RooRarFit/doc/tutorial/*. You will first need to generate the test dataset. To generate a dataset that can be used for regression testing or tutorials:

- Make sure $ROOTSYS is defined
- Create a subdirectory ./Ntuples
- Run the command:

```
> $ROOTSYS/bin/root -b -q -l RooRarFit/doc/tutorial/make_data.C |& tee make_data.t
```

This will create 3 datasets (in ascii and root format) in ./Ntuples

- *Ntuples/tut_signal.{dat,root}* has 10000 Signal events.
- *Ntuples/tut_uds.{dat,root}* has 10000 Continuum events.
- *Ntuples/tut_bkg.{dat,root}* has 2000 Peaking background events.

The dataset is a simulated B0/B0bar (or B+/B-) decay at the B-factories including mixing and CP violation. The dataset can be used to measure yields, branching fractions, charge CP asymmetries and Time-dependent CP Violation with event-by-event errors.

The datasets contain the following variables (in this order in the ascii files):

Variable	**Description of the input variables**
mes	B mass (GeV/c^2).
deltae	Difference in energy of B and sqrt(s)/2 (GeV).
mass	A resonance mass (GeV/c^2).
nn	A Multi-Variate distribution (e.g. Fisher).
dt	Time difference between the two B decays (ps).
tag	Flavour tag (B0=1,B0bar=-1) if data treated as B0/B0bar decay.
sigmode	a flag that indicates if the K* decayed to K+pi- or K0s pi0. This can be used as a test of Simultaneous fits and branching fraction measurements.
charge	The charge (B+=+1,B-=-1) if data treated as B+/B- decay. The signal has a charge asymmetry Acp = -0.05.
run	A run number (500 or 1000 = signal, 2000 = continuum bkg, 3000 = peaking bkg).
dterr	The error on the time difference, *dt* (ps).

Once the dataset has been generated the configuration files can be used to test various aspects of RooRarFit. The configuration file will create a test "real dataset" that will contain: 100 signal events, 4000 continuum background events and 500 peaking background events. The input values of the fitted variables and the values returned by RooRarFit are given in the following table:

Variable	**Input**	**Fitted**
Yield (events)	100	105 +/- 18

BF (x 10^-6)	5.22	5.47 +/- 1.0
Charge Asymmetry	-0.05	-0.19 +/- 0.16
S	0.7	0.6 +/- 0.22
C	0.0	0.003 +/- 0.15

If you want more accurate fitted results, increase the number of signal events in the "real data" sample in *tutorial.config* e.g.

```
// In tutorial.config, edit the next line to increase the number of
// signal events sigMC from 100 (up to 10000)
simData  = add "MC cocktail" sigMC 100 bkgMC 500 udsMC 4000
```

When you run the examples, the numerical values of the fits are stored in the *./.params* directory and the ROOT files containing any plots, ntuples or histograms are stored in *./results*.

3.2 Example 1 (Yields and Actions):

- To fit the Pdfs for a signal yield measurement (initial values of the parameters are created in *tutorial.config*):

```
> rarFit -A PdfAct doc/tutorial/tutorial.config
```

- To extract the yield from the simulated "real" dataset (initial values of the parameters are taken from the output of PdfAct):

```
> rarFit -A MLAct doc/tutorial/tutorial.config
```

- To do a pure toy study on an ensemble of events created by the fitted PDFs (initial values of the parameters are taken from the output of MLAct):

```
> rarFit -A ToyAct doc/tutorial/tutorial.config
```

- To do an embedded toy study on an ensemble of events created from the datasets (initial values of the parameters are taken from the output of MLAct):

```
> rarFit -A eToyAct doc/tutorial/tutorial.config
```

- To generate projection plots of the yield (values of the parameters are taken from the output of MLAct):

```
> rarFit -A ProjAct doc/tutorial/tutorial.config
```

- To generate SPlot plots of the yield (values of the parameters are taken from the output of MLAct):

```
> rarFit -A SPlotAct doc/tutorial/tutorial.config
```

- To generate a scan of -log(L) around the yield minimum (values of the parameters are taken from the output of MLAct):

```
> rarFit -A YieldScan doc/tutorial/tutorial.config
```

3.3 Example 2 (Branching Fraction):

To convert the signal yield into a branching fraction taking into account any fit bias:

- To fit the individual PDFs:

```
> rarFit -A PdfAct -C "bfFitter Config" doc/tutorial/tutorial.config
```

- To extract the branching fraction from a simulated "real" dataset

```
> rarFit -A MLAct -C "bfFitter Config" doc/tutorial/tutorial.config
```

- Do a "pure" toy study on an ensemble of events created by the fitted PDFs

```
> rarFit -A ToyAct -C "bfFitter Config" doc/tutorial/tutorial.config
```

- Do a "embedded" toy study on an ensemble of events created from the datasets

```
> rarFit -A eToyAct -C "bfFitter Config" doc/tutorial/tutorial.config
```

- To generate a scan of -log(L) around the branching fraction minimum

```
> rarFit -A BRScan doc/tutorial/tutorial.config
```

3.4 Example 3 (Charged asymmetries):

To fit the PDFs for a charged asymmetry Acp measurement, first change *"simultaneousFit = no"* to *"simultaneousFit = yes"* in section *"[mlFitter Config]"* in *tutorial.config*. Then:

- To fit the individual PDFs

```
> rarFit -A PdfAct doc/tutorial/tutorial.config
```

- To extract the charge asymmetries from the simulated "real" dataset

```
> rarFit -A MLAct tutorial.config
```

- To generate a scan of -log(L) around the charge asymmetry minimum

```
> rarFit -A AScan tutorial.config
```

3.5 Example 4 (time dependent asymmetries)

To fit the PDFs for a time dependent measurement,

- To fit the individual PDFs

```
> rarFit -A PdfAct doc/tutorial/tutorial_v3.config
```

- To extract the asymmetries C and S from the simulated "real" dataset

```
> rarFit -A MLAct doc/tutorial/tutorial_v3.config
```

4 Questions and Answers

1. How to build this documentation?

   You will need versions of `texinfo`, `doxygen` and `tex` to build some or all the documentation

   ```
   > cd RooRarFit/doc
   # make the pdf/postscript version of the user guide
   > make RooRarFitMakeInfo
   # make the html version of the user guide
   > make RooRarFitTexinfo
   # make the dOxygen documentation of the source code
   > make RooRarFitDoxygen
   ```

2. My splitting rule for a model is very long, is there any limits on it?

   `RooFit` does have limit on how long a splitting rule can go for a model. (The limitation will be removed in the next major `RooFit` release.) The current default value is 4096.

3. How to blind yields?

   CP parameters `C` and `S` are the variables can be set to blind easily. You can choose to blind your yields (or any variable), but it is not quite as easy.

   ```
   [yourYieldModel Config]
   yBlindCat = yBlindCat RooCategory "Yield blind state" useIdx Blind 1 Unblind 0
   Comps  = Sig Chmls Bkg
   Coeffs = nbSig nbChmls nbBkg
   nbSig = nbSig RooUnblindOffset "blind nSig" "blind yield of mode xxx" 10 nSig yBli
   nbChmls = nbChmls RooUnblindOffset "blind nChmls" "blind yield of mode xxx" 10 nCh
   nbBkg = nbBkg RooUnblindOffset "blind nBkg" "blind yield of mode xxx" 20 nBkg yBli
   nSig   = nSig    200 L(10 - 500)
   nChmls = nChmls 1000 L(0 - 10000)
   nBkg   = nBkg   2000 L(0 - 10000)
   ```

   You may also want to add option `"b"` to configuration `mlFitOption` in action section to suppress RooFit messages from fit.

4. I would like to include a component in my fit for self crossfeed (SXF) signal events. I will then fix the fraction of self crossfeed, and then in my final fit just float the total signal yield.

   Sample configs dealing with SXF Pdf in that fashion can be found in sxfSig.config (`http://rarfit.sourceforge.net/Sample_configs/sxfSig.config`). Please notice that this configuration file is NOT complete.

   - I would like to see also the 'total' signal PDF, i.e. the sum of the two components, fit to the complete dataset.

     Those plots as projection of the two components are given by default, however, you should not fit the full dataset on the combined pdf, otherwise we end up with using the old pdf setting without SXF separation.

5. I am trying to set up a toy job where I embed signal MC and BB MC events with continuum background events generated from the PDFs and then fit with only the

signal and continuum background components (no BB component in the fit). I want to use this type of fit to determine if I need a BB component or not.

Suppose your fitter has been setup correctly with two yield components, `nSig` and `nBkg`, your signal MC dataset is `sigMC`, and BB dataset is `bbMC`. In your action section for toy study, change or add:

```
toySrc_nSig = sigMC 90 // 90 embedded signal events for nSig
toySrc_nBkg = bbMC 27 pdf "@0-27 nBkg" // Bkg has two parts:
                                       // 27 from bbMC, the rest from pdf
```

6. Why my pulls for embedded yields look wrong?

   See Section 2.3.2 [toyStudy Action], page 48, (2nd paragraph).

7. How to get number of embedded events from individual samples in toy study output `root` file?

   Entries for number of embedded events from individual samples in toy study `root` file are named as `embdEvt_<toySrc_name>`, for example, embedded event number for `sigMC` is in entry named `embdEvt_sigMC`.

8. I have deSig pdf parameters split, how can I get the split parameters for simultaneous pdf in the fit?

   In `[deSig Config]` section, add/change

   ```
   pdfFit = simFit
   ```

   See Section 2.2.1 [commonPdfConfig], page 17.

9. How to fit components of product pdf together?

   For example, `SigPdf` is product PDF, in section `[SigPdf Config]` add/change config

   ```
   ndFit = yes
   ```

10. How to create/add my own `RooAbsPdf`?

    See Section 2.2.31 [rarUsrPdf Configuration], page 45.

11. I have built a model. Can I fix/float on fit action basis some parameters previous floated/fixed when I built it in PDF configuration sections?

    Yes. For example, if you have `nSig` as yield and float it, but want to fix it for sPlot, in action section for sPlot, add/change:

    ```
    preMLFix = nSig
    ```

    Please go through configs `postPdfFloat`, `preMLFix`, and `preMLFloat` for more detailed instructions.

12. Can I embed a fraction of event, for example, 11.3, for embedded toys? This does not make sense to me since you can only get integer events from datasets.

    Yes, you can. Here is how it works. If you allow Poisson fluctuation, it will embed number of events per experiment which is subject to Poisson fluctuation with mean at 11.3. If you do not allow Poisson fluctuation, it will generate 11 or 12 events randomly per toy experiment with mean at 11.3.

13. Can I use toy sample for mlFit, scanPlot, sPlot, etc?

    Yes. For `toyStudy`, the first toy sample generated will be saved as `"toySample"`, so in the same action section, other actions following `toyStudy` can use the sample, for example, to do `sPlot`. Just give `toySample` as configuration for input data, for example

```
[ToySPlotAct]
// toy options
toyStudy = yes
preToyReadParams = A mlFit
preToyReadSecParams = MCParamSec
toyNexp = 1 // # experiments
toyNevt = 0 extended
// init values
nKKKst = 70  +/- 44 C L(-1000 - 1000)
nKsKsKs = 20  +/- 44 C L(-1000 - 1000)
nKKPiZ = 2.3 +/- 100 C L(0 - 1000)
nSig = 100 +/- 20.00 L(-100 - 500)
// for embedded toy study
toySrc_nSig = sigMC "@0 nSig"

[useToySampleMLFit]
// ml fit
mlFit = yes
mlFitData = toySample
preMLReadParams = no
preMLReadSecParams = no
postMLWriteParams = no

[useToySampleSPlot]
// sPlot
sPlot = yes
sPlotData = toySample
sPlotSaveSWeight = yes
sPlotVars = mes de
preSPlotReadParams = no
preSPlotReadSecParams = no
```

Index

(Index is nonexistent)