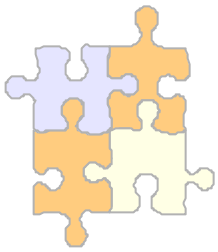


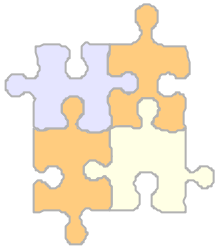
RooFit Tutorial – Managing complex fits

Wouter Verkerke (UC Santa Barbara)
David Kirkby (UC Irvine)

Overview

- WARNING - This tutorial is incomplete
  - Relevant parts of the retired ‘advanced’ tutorial have beenmoved here.
- Scope of this tutorial
  - Issues arising in building and managing a large scale fit project in RooFit
  - Use sin2β fit frame work as illustration: Focus on topics such as
    - Blinding
    - Managing and building a large number of similar PDFs
    - How to deal with per-event errors
    - Non-trivial plots (e.g. projecting out per-event errors)
    - Setting up ToyMC studies for e.g. goodness-of-fit determination
    - Writing new PDF classes

Managing many PDFs



The sin2β fit as example

Automating PDF replication

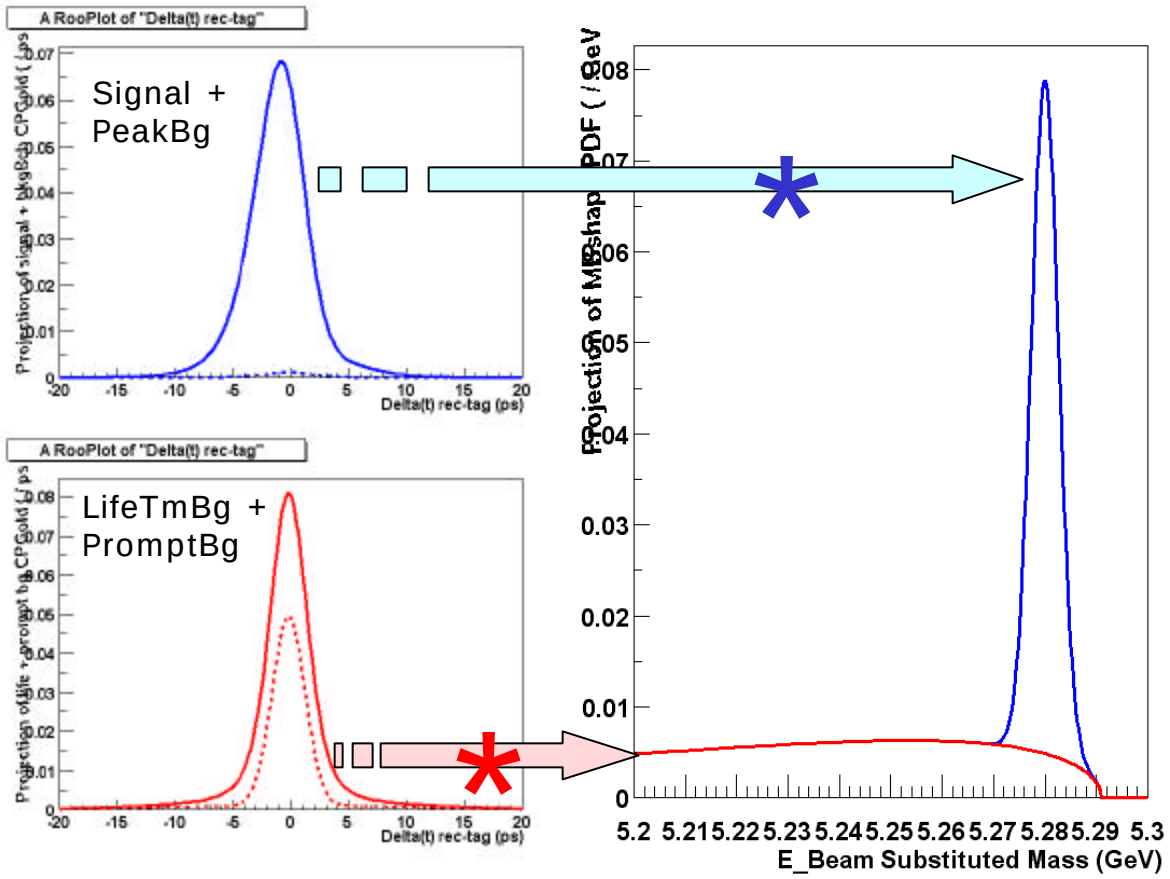
RooSimPdfBuilder

Wouter Verkerke, UCSB

The sin2beta fit model in two slides

- Simultaneous fit of
  - CP data to CP model and BReco data to mixing model
  - Split data by tagging category and CP event type (Gold,Klong[IFR/EMC][ee/μμ])

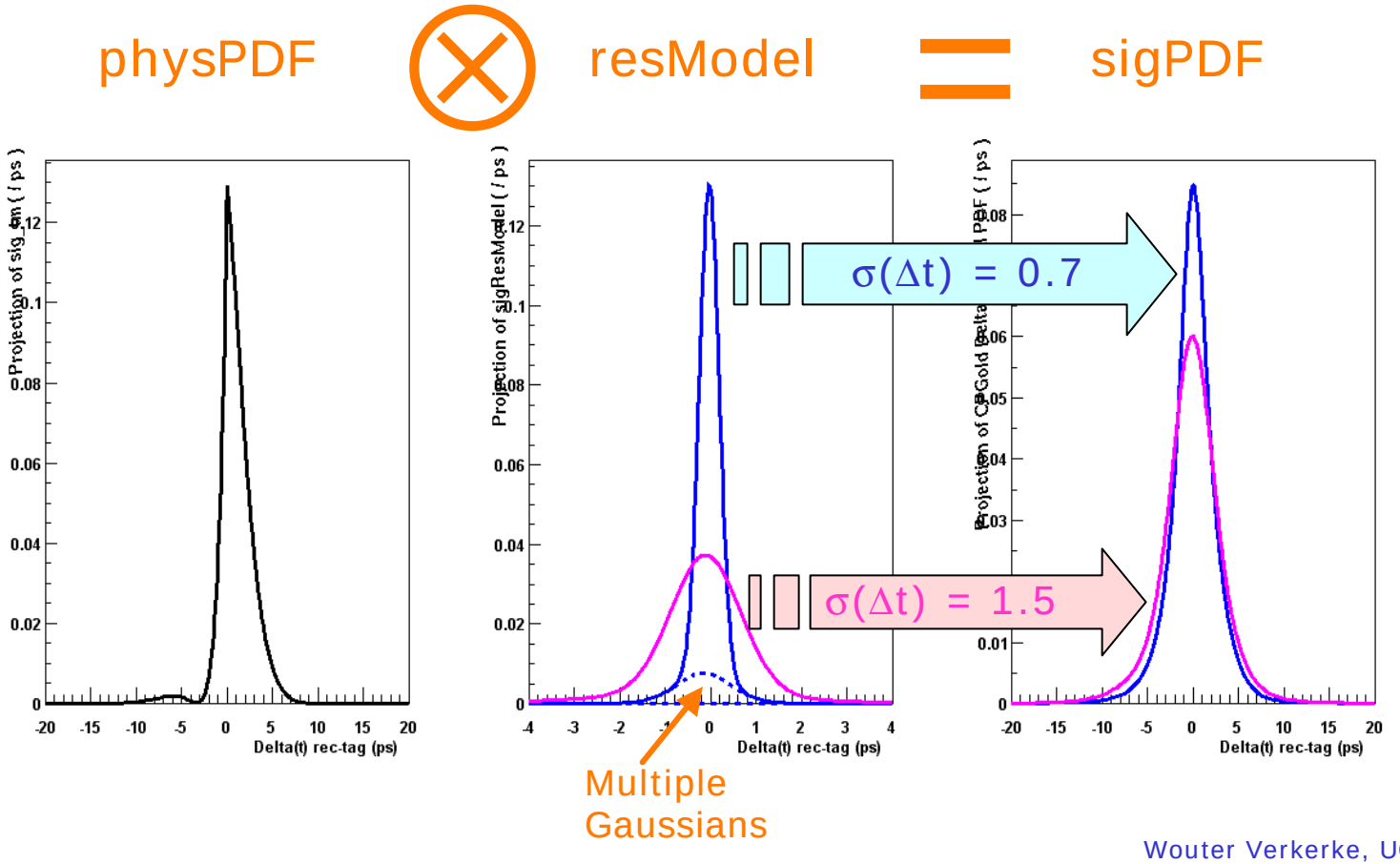
Dt model * **mB model**



CP-gold PDF and Mixing PDF have same structure

The sin2beta fit model in two slides

- Each Δt PDF is a convolution of a physics PDF with a 2 or 3 gauss resolution model.
  - The bias and width of the resolution model are scaled with the per-event error



Building complex simultaneous fits

- The essence of the sin2β fit is simple
  - For each physics event type (CP-gold/CP-klong/Breco-mix) there is one 'prototype' PDF
  - Simultaneous fit would look like this

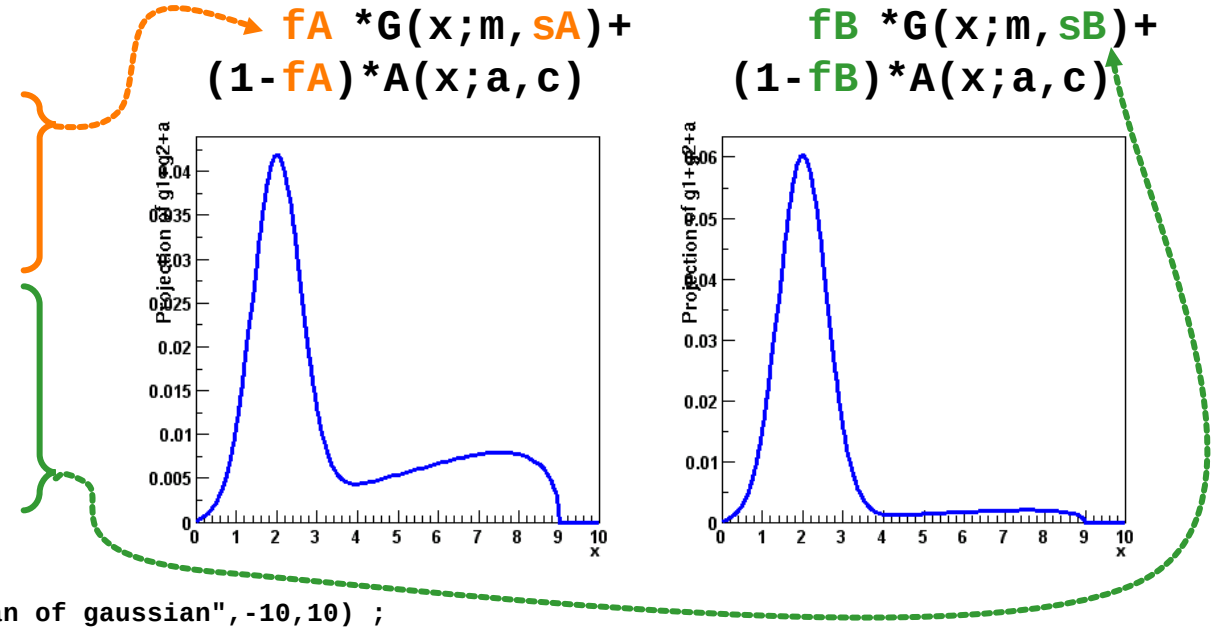
```
// Build Simultaneous CP/Mixing PDF
RooSimultaneous simPdf("simPdf","CP/mixing PDF",physCat) ;
simPdf.addPdf(cpgoldPDF,"CP-gold") ;
simPdf.addPdf(klongPDF,"CP-klong") ;
simPdf.addPdf(mixingPDF,"BR-mix") ;

// Perform the fit
simPDF.fitTo(data,"fit options")
```

RooCategory in dataset defining event physics type

- *Complications arise from further subdivision of data*
  - Data is subdivided by tagCat, KL-reco type etc
  - Need to create many PDFs that trivially differ from each other:
    - CP-goldPdf_Kao, CP-goldPdf_Lep, CP-goldPdf_NT1, CP-goldPdf_NT2 CP-klongPdf_Kao_EMC, CP-klongPdf_Kao_IFR, CP-klongPdf_Lep_EMC,...
  - Lots of bookkeeping involved

PDF replication: manual approach

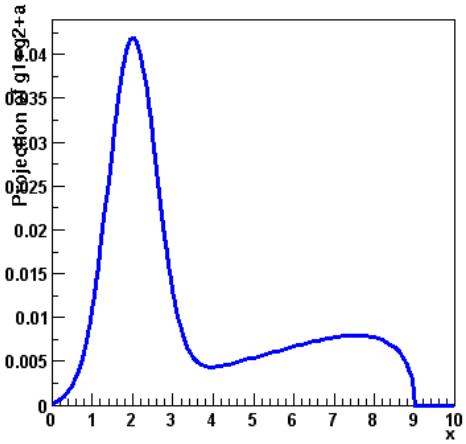
x	type
0.73	A
0.42	A
0.33	A
1.52	B
0.29	B
0.98	B
0.54	B




```
RooRealVar m("m","mean of gaussian",-10,10) ;
RooRealVar s_A("s_A","sigma of gaussian A",0,20) ;
RooGaussian gauss_A("gauss_A","gaussian A",X,m,s_A) ;
RooRealVar s_B("s_B","sigma of gaussian B",0,20) ;
RooGaussian gauss_B("gauss_B","gaussian B",X,m,s_B) ;

RooRealVar k ("k","ArgusBG kappa parameter",-50,0) ;
RooRealVar xm("xm","ArgusBG cutoff point",5.29) ;
RooArgusBG argus ("argus","argus background",X,k,xm) ;

RooRealVar f_A("f_A","fraction of gaussian A",0.,1.) ;
RooAddPdf pdf_A("pdf_A","gauss_A+argus",RooArgList(gauss_A,argus_A),f_A) ;
RooRealVar f_B("f_A","fraction of gaussian A",0.,1.) ;
RooAddPdf pdf_B("pdf_B","gauss_B+argus",RooArgList(gauss_B,argus_B),f_B) ;

RooSimultaneous simPdf("simPdf","simPdf",type) ;
simPdf.addPdf(pdf_A,"A") ;
simPdf.addPdf(pdf_B,"B") ;
```

PDF replication: automated approach

f *G(x;m,s)+
(1-f)*A(x;a,c)



Prototype PDF

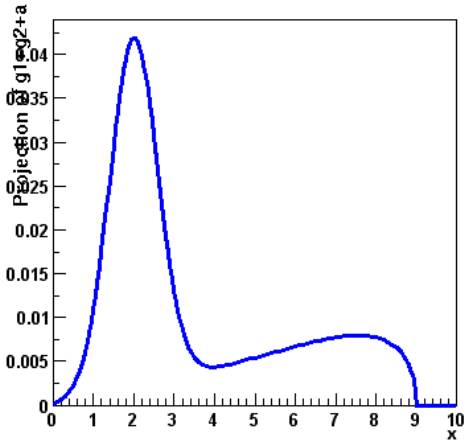
```
RooRealVar m("m","mean",-10,10) ;
RooRealVar s("s","sigma",0,20) ;
RooGaussian gauss("gauss","g",X,m,s);

RooRealVar k("k","kappa",-50,0) ;
RooRealVar xm("xm",“ebeam",5.29) ;
RooArgusBG argus("argus","a",X,k,xm);

RooRealVar f(“f","f(gauss)",0.,1.) ;
RooAddPdf pdf("pdf","gauss+argus",
     RooArgList(gauss,argus),gfrac) ;
```

```
RooSimPdfBuilder builder(pdf) ;
RooArgSet* config = builder.createProtoBuildConfig() ;
(*config)["physModels"] = "pdf" ;
(*config)["splitCats"]  = “type" ;
(*config)["pdf"]        = “type : f,s" ;
RooSimultaneous* simPdf = builder.buildPdf(*config,&D) ;
```

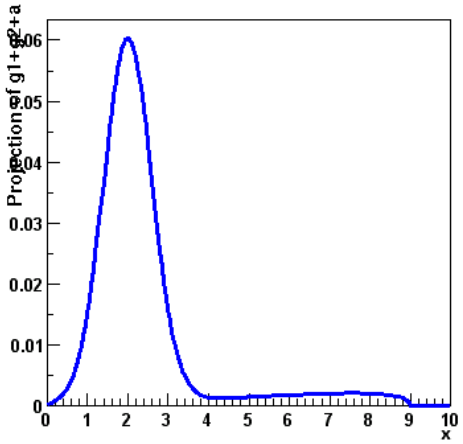
Customization prescription

```
f ® f[type]
s ® s[type]
```

fA *G(x;m,sA)+
(1-fA)*A(x;a,c)



fB *G(x;m,sA)+
(1-fB)*A(x;a,c)



Customized PDF copies

PDF replication: **RooSimPdfBuilder**

- **RooSimPdfBuilder** is a factory class
  - takes a collection of prototype PDFs
  - builds a **RooSimultaneous** given a set of parameter splitting prescriptions
  - Configuration supplied by a **RooArgSet** of **RooStringVar**s
    - Can be filled in a macro, or read from a file
- Simple build: One prototype PDF

Specifies which prototype will be used in this build

```
// prototype is gauss+argus of previous example
physModel = pdf
splitCat  = type
pdf       = type : f,s
```

Specifies which categories of the dataset will be used to define data subsets

Specifies how prototype 'pdf' should be tailored for each data subset: *Parameters **f** and **s** should individual per state of **type***

PDF replication: **RooSimPdfBuilder**

```
// Build the simultanous PDF
RooSimultaneous* simPdf = builder.buildPdf(config,data) ;
RooSimPdfBuilder::buildPdf: list of physics models (pdf)
RooSimPdfBuilder::buildPdf: list of splitting categories (type)
RooSimPdfBuilder::buildPdf: processing physics model pdf
RooSimPdfBuilder::buildPdf: configured customizers for all physics models
RooCustomizer for sum
  Splitting rules:
   f is split by type
   s is split by type
RooSimPdfBuilder::buildPdf: Customizing physics model sum for mode {A}
RooSimPdfBuilder::buildPdf: Customizing physics model sum for mode {B}

// Print the parameters of the built PDF
simPdf->getParameters(data)->Print("v") ;
RooArgSet::parameters:
  1) RooRealVar::k    : -1.00000 C
  2) RooRealVar::xm   :  9.0000  C
  3) RooRealVar::f_A  :  0.50000 C
  4) RooRealVar::f_B  :  0.50000 C
  5) RooRealVar::m    :  2.0000  C
  6) RooRealVar::s_A  :  0.60000 C
  7) RooRealVar::s_B  :  0.60000 C
```

Individual by **type**, as specified

Individual by **type**, as specified

PDF replication: **RooSimPdfBuilder**

- Build with >1 prototype
  - To be used when prototype itself RooSimultaneous (e.g. sin2β)

Proto-**RooSimultaneous** prescription:
- Index category is **physCat**
- Associate **physCat** state **CPGold** with PDF **GoldPDF**
- Associate **physCat** state **BMix** with PDF **BRMixPDF**

Split data in subsets of
tagCat ´ **runBlock** (´ **physCat**)

```
physModel = physCat : CPGold=CPGoldPDF Bmix=BRMixPDF
splitCat  = tagCat runBlock
CPGoldPdf = tagCat           : x,y,z,… \\
            runBlock         : foo     \\
            tagCat,runBlock  : zaza
BRMixPDF  = tagCat           : x,a,b   \\
            runBlock         : bar
```

Specifies how prototype ‘CPGoldPdf’ should be tailored:
- *Parameters* ***x,y,z*** *should individual per state of* ***tagCat***
- *Parameter* **foo** *should be individual per state of* **runBlock**
- *Parameter* **zaza** *should be individual per state of* **tagCat** *and* **runBlock**

Specifies how prototype ‘BRMixPDF’ should be tailored

PDF replication: **RooSimPdfBuilder**

- Using the same prototype for multiple **physCat** states
  - Split parameters by **physCat** to distinguish PDFs

The same prototype is assigned to 2 physCat states

```
physModel = physCat : KlEmc=KlongPDF KlIfr=KLongPDF …
splitCat  = tagCat runBlock
KlongPdf  = tagCat          : x,y,z,… \\
            physCat         : foo
            physCat,tagCat  : bar
```

The parameter **foo** will be individual for **physCat=KlEmc** and **physCat=KlIfr**, distinguishing the PDF for the two **physCat** states

```
foo_KlEmc
foo_KlIfr
bar_{KlEmc,Kao}
bar_{KlIFR,Kao
…
```

PDF replication: **RooSimPdfBuilder**

- Non-trivial parameters splits
  - Sofar only 'simple' splits were used (grid structure)

tagCat

Lep
Kao
NT1
NT2

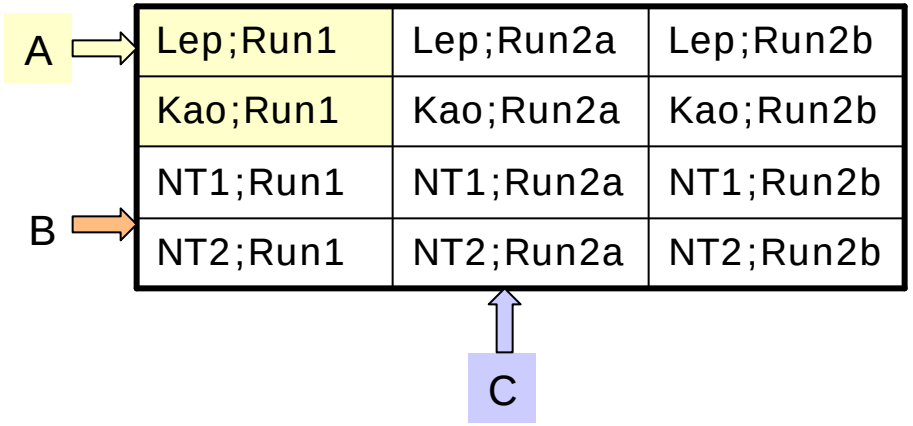
runBlock

Run1	Run2a	Run2b

tagCat ´ runBlock

Lep;Run1	Lep;Run2a	Lep;Run2b
Kao;Run1	Kao;Run2a	Kao;Run2b
NT1;Run1	NT1;Run2a	NT1;Run2b
NT2;Run1	NT2;Run2a	NT2;Run2b

- Q: How do split a parameter in regions {A,B,C}?



A: Define a category function that implements the transformation (tagCat,runBlock) → {A,B,C}

Options:
- **RooGenericCategory**
- **RooMappedCategory**

PDF replication: **RooSimPdfBuilder**

- Given a category function **superCat** as function of **tagCat,runBlock**
- If splitting *only* in **superCat** and not in **tagCat** or **runBlock**
  - Add **superCat** as precalculated column to the dataset

```
data->addColumn(superCat) ;
```

  - Proceed as usual, specifying **superCat** as splitting category

```
physModel = pdf
splitCat  = superCat
CPGoldPdf = superCat          : zaza
```

- If splitting in *both* **superCat** *and* it's ingredients (**tagCat** etc)
  - Must take correlations into account, specify superCat as auxiliary splitting category

```
RooSimultaneous* simPdf =
          builder.buildPdf(config,data,superCat) ;
```

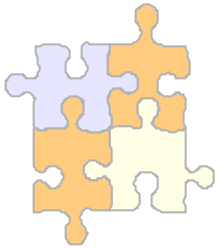
```
physModel = pdf
splitCat  = tagCat runBlock
CPGoldPdf = tagCat            : x,y,z,… \\
            runBlock          : foo     \\
            superCat          : zaza
```

PDF replication: putting it together for sin2β

- Using **RooSimPdfBuilder** the implementation of the sin2β fit is easy.
  - Given the prototype PDFs (CPGold,CPKlong,Bmixing), the following builder configuration will build the *full* PDF!

```
physModels    = physCat           : Gold=CPGold BMix=BMixing
                                    KlEmcE=KLong KlEmcM=KLong KlIfrE=KLong KlIfrM=KLong
splitCats     = tagCat
BMixing       = tagCat            : sig_eta,sig_deta,bgC_eta,bgL_eta,
                                    bgP_eta,sigC_bias,outl_bias
                physCat,tagCat    : bgP_f,mbMean,mbWidth,argPar,sigfrac
                physCat           : bgC_f
CPGold        = tagCat            : sig_eta,sig_deta,sigC_bias,outl_bias
                physCat           : mbMean,mbWidth,argPar,bgP_f,bgC_f
                physCat,tagCat    : sigfrac
KLong         = tagCat            : sig_eta,sig_deta,sigC_bias,outl_bias
                physCat,tagCatL   : klSig_frac,klBgCcK_frac,klKstBg_frac,klKshBg_frac,
                                    klNPLBg_frac,klNPPBg_frac,klPsxBg_frac
                physCat           : deGMeanSig,deGWidthSig,deACutoffSig,deAKappaSig,
                                    deGFracSig,dePol1IPbg,dePol2IPbg,dePol3IPbg,dePol4IPbg,
                                    deACutoffSBbg,deAKappaSBbg,deGMeanKSBg,deGWidthKSBg,
                                    deACutoffKSBg,deAKappaKSBg,deGFracKSBg,deG2MeanSig,
                                    deG2WidthSig,deG2FracSig,psix_cpev
```

Configuring your fit



How to manage your configuration data

Wouter Verkerke, UCSB

Configuration data

- Complex fits such as sin2β need lots of configuration data
  - Collection of input datasets
  - Structural definition of PDF
  - Blinding string / configuration
  - Initial parameters
- Use standard RooFit classes to store your configuration data
  - Each configuration item can be represented by a
    - **`RooRealVar`** – Fit parameters etc
    - **`RooCategory`** – Switches, options etc
    - **`RooStringVar`** – File names, blinding strings etc
  - A set of configuration values is represented by a **`RooArgSet`**
  - **`RooArgSet`** provides methods to write its contents to a human-readable ASCII file and to read it back

Configuration data

```
// Read input data section
RooStringVar charmDir("charmDir","charmDir","") ;
RooStringVar charmFiles("charmFiles","charmFiles","") ;
RooArgSet(charmDir,charmFile).readFromFile("config.txt",0,"Input Files") ;
```

Read file one section at a time

All configuration data labeled by name

config.txt

```
[Input Data]

charmDir   = /nfs/farm/babar/AWG/sin2b/data_run2/
charmFiles = dt_JpsiKs_run1.fit,dt_JpsiKs_run2.fit

[Blinding]

  blindingStatus = Blind
  blindingString = No kstar in this fit
  blindingSigma  = 0.1
  blindingMPoint = 0.6

[CP/mixing-fit structure]

  physModels   = physCat          : Gold=CPGold BMix=BMixing
  splitCats    = tagCat
  CPGold       = tagCat           : sig_eta,sig_deta,sigC_bias
                 physCat          : mbMean,mbWidth,argPar,bgP_f
                 physCat,tagCat   : sigfrac

[Initial parameter values]

  argPar_Gold        = -24.42 L(-100 - 0)
  argPar_{BMix;Kao}  = -27.88 L(-100 - 0)
  argPar_{BMix;Lep}  = -41.53 L(-100 - 0)
  argPar_{BMix;NT1}  = -33.86 L(-100 - 0)
  argPar_{BMix;NT2}  = -36.64 L(-100 - 0)
  mbMax              =  5.291 C
```

Configuration data

RooSimPdfBuilder configuration is already a **RooArgSet** of **RooStringVar**s

config.txt

```
[Input Data]

  charmDir   = /nfs/farm/babar/AWG/sin2b/data_run2/
  charmFiles = dt_JpsiKs_run1.fit,dt_JpsiKs_run2.fit

[Blinding]

  blindingStatus = Blind
  blindingString = No kstar in this fit
  blindingSigma  = 0.1
  blindingMPoint = 0.6

[CP/mixing-fit structure]

  physModels   = physCat          : Gold=CPGold BMix=BMixing
  splitCats    = tagCat
  CPGold       = tagCat           : sig_eta,sig_deta,sigC_bias
                 physCat          : mbMean,mbWidth,argPar,bgP_f
                 physCat,tagCat   : sigfrac

[Initial parameter values]

  argPar_Gold        = -24.42 L(-100 - 0)
  argPar_{BMix;Kao}  = -27.88 L(-100 - 0)
  argPar_{BMix;Lep}  = -41.53 L(-100 - 0)
  argPar_{BMix;NT1}  = -33.86 L(-100 - 0)
  argPar_{BMix;NT2}  = -36.64 L(-100 - 0)
  mbMax              =  5.291 C
```

Configuration data

A **RooArgSet** with all the parameters of a PDF can trivially be obtained

```
RooArgSet* allParams = pdf->getParameters(data) ;
allParams->readFromFile("config.txt","READ","Initial parameter values") ;
```

Flag all initialized parameters

Automatically list parameters *not* initialized from file

```
allParams->selectByAttrib("READ",kFALSE)->Print("v") ;
```

Contents of list automatically includes all split parameters introduced by **RooSimPdfBuilder**

config.txt

```
[Input Data]

  charmDir   = /nfs/farm/babar/AWG/sin2b/data_run2/
  charmFiles = dt_JpsiKs_run1.fit,dt_JpsiKs_run2.fit

[Blinding]

  blindingStatus = Blind
  blindingString = No kstar in this fit
  blindingSigma  = 0.1
  blindingMPoint = 0.6

[CP/mixing-fit structure]

                                                  Gold=CPGold BMix=BMixing

                                                  sig_eta,sig_deta,sigC_bias
                    physCat                     : mbMean,mbWidth,argPar,bgP_f
                    physCat,tagCat              : sigfrac

[Initial parameter values]

  argPar_Gold          = -24.42 L(-100 - 0)
  argPar_{BMix;Kao}    = -27.88 L(-100 - 0)
  argPar_{BMix;Lep}    = -41.53 L(-100 - 0)
  argPar_{BMix;NT1}    = -33.86 L(-100 - 0)
  argPar_{BMix;NT2}    = -36.64 L(-100 - 0)
  mbMax                =  5.291 C
```

Configuration data, advanced options

- Limited *pre-processing* available

BaBar-style and C++ style inline comments allowed

Echo command prints user messsages while reading the configuration

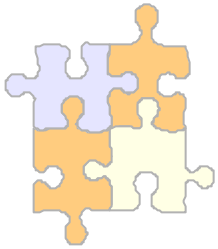
Include other configuration files Recursion allowed

Conditionals with full C++ syntax. All variables in the read **RooArgSet** can be referenced in the expression. Nested conditionals OK

Abort statement forces RooArgSet::read to fail with error states

```
# coments BaBar style
// comments C++ style
file = foo.txt // comments C++ style

[Initial parameter values]
  echo Now processing parameter section
  argPar_Gold         = -24.42 L(-100 - 0)
  argPar_{BMix;Kao}   = -27.88 L(-100 - 0)
  argPar_{BMix;Lep}   = -41.53 L(-100 - 0)
  argPar_{BMix;NT1}   = -33.86 L(-100 - 0)
  argPar_{BMix;NT2}   = -36.64 L(-100 - 0)

  include common_parameters.txt

  if (runMode==runMode::ExtraFancy)
     fancyPar = 5.0 +/- 0.3 L(0-10)
  else if (runMode==runMode::Normal)
     normalPar = 17.0 L(10-20)
  else

     echo You can do this
     abort

  endif
```

Blinding

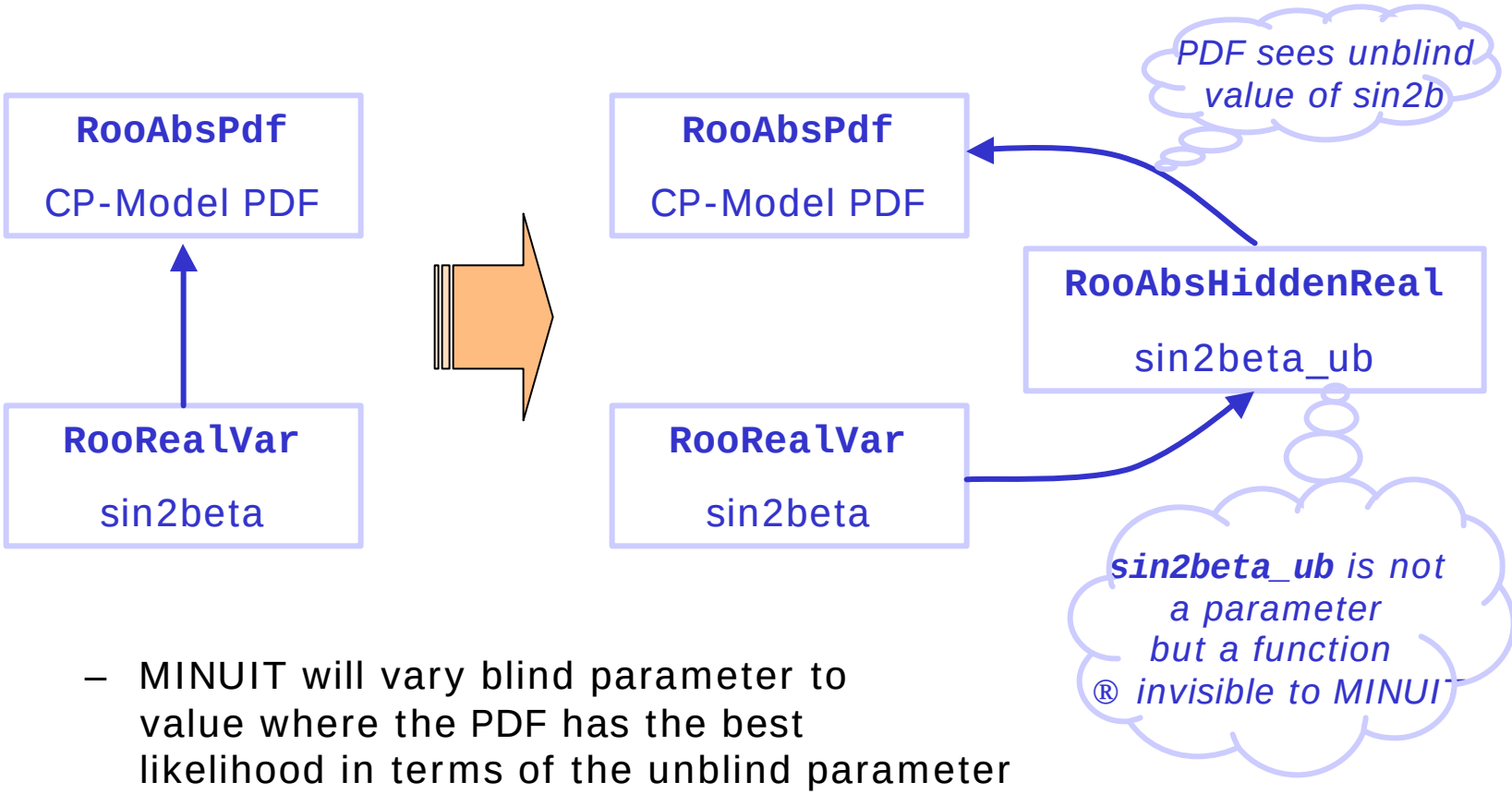


Blinding fit parameters

Wouter Verkerke, UCSB

Parameter blinding

- What does blinding a parameter involve?
  - Insert unblinding transformation between parameter value holder and PDF object using that parameter value



  - MINUIT will vary blind parameter to value where the PDF has the best likelihood in terms of the unblind parameter

Parameter blinding

- Blinding-related classes in RooFitModels
  - **RooBlindTools**
    - replica of BaBar **BlindTools** class
  - **RooUnblindPrecision,**
    RooUnblindOffset
    RooUnblindUniform
    RooUnblindCPDeltaT
    RooUnblindCPAsym

Implementations of **RooAbsHiddenReal** wrapping 5 different blinding methods from RooBlindTools

Each method can take an optional **RooCategory** parameter that serves as 'blinding switch' → Disable blinding without structural change

- How to keep secrets
  - The purpose of blinding is to avoid *accidental* exposure of the blinded parameter
    - RooFit is a toolkit, not a monolithic black box
  - Classes derived from **RooAbsHiddenReal** will *never* show their value in **Print()** and other display routines
  - Unblinder functions never show up in any parameter list
  - The **RooAbsHiddenReal::getVal()** method is **protect-**ed. Cannot be called from the ROOT command line without explicit cast
  - To intentionally unblind call **RooAbsHiddenReal::getHiddenVal()**

Parameter blinding

- Example implementation

```
// Sin2beta variable
sin2b = new RooRealVar("sin2b","sin(2*beta)",-1.0,4.0) ;

// Blinding switch [ category parameter ]
s2b_bs   = new RooCategory("s2b_bs","sin2beta blinding state") ;
s2b_bs->defineType("Unblind",0) ;
s2b_bs->defineType("Blind",1) ;

// Unblinding transformation
sin2b_ub = new RooUnblindPrecision("sin2b_ub","Unblinded Sin2beta",
                                   _blindString,_cval,_sigma,
                                   *sin2b,*s2b_bs,kTRUE) ;

// Use sin2b_ub in fit where ever sin2beta input is needed

// Activate/Deactivate blinding at any moment
s2b_bs->setLabel("Blind"/"Unblind") ;
```