

Status of the Common PWA-Framework for PANDA

Mathias Michel
Helmholtz Institut Mainz



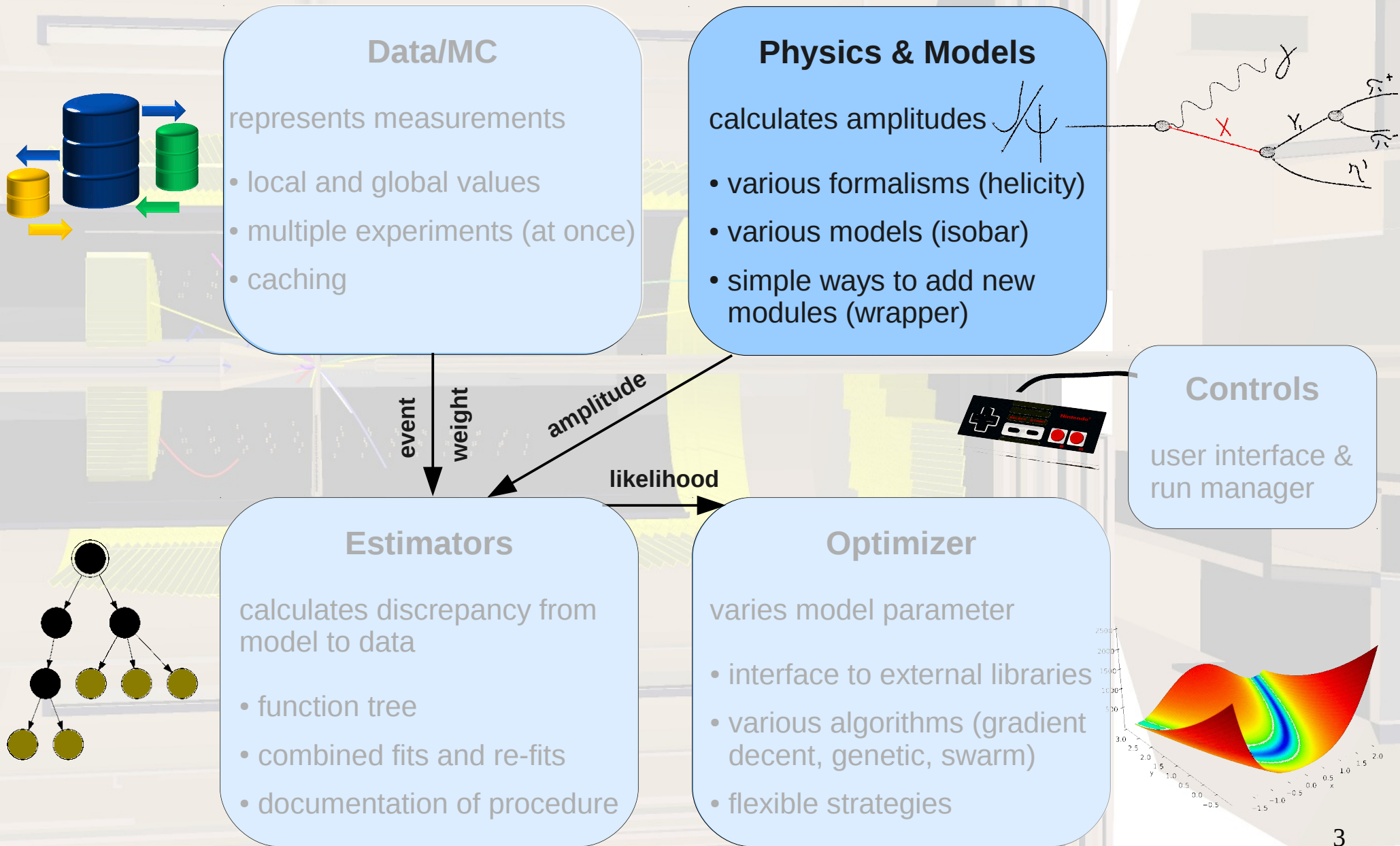
Panda CM, GSI
10.12.2012



Overview

- Requirements/Idea of Framework
- Infrastructure
- Modules
- First Fit and Tests
- Status

PWA Framework



PWA Framework

```
graph TD; DataMC[Data/MC] -- "event" --> Estimators; DataMC -- "weight" --> Estimators; Physics[Physics & Models] -- "amplitude" --> Estimators; Estimators -- "likelihood" --> Optimizer; Physics -- "likelihood" --> Optimizer; Controls[Controls] --- Optimizer;
```

Data/MC

represents measurements

- local and global values
- multiple experiments (at once)
- caching

Physics & Models

calculates amplitudes

- various formalisms (helicity)
- various models (isobar)
- simple ways to add new modules (wrapper)

Estimators

calculates discrepancy from model to data

- function tree
- combined fits and re-fits
- documentation of procedure

Optimizer

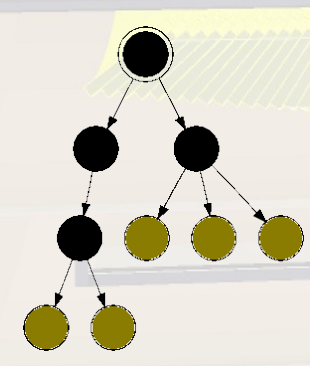
varies model parameter

- interface to external libraries
- various algorithms (gradient decent, genetic, swarm)
- flexible strategies

Controls

user interface & run manager

The diagram illustrates the PWA Framework architecture. It consists of five main components: Data/MC, Physics & Models, Estimators, Optimizer, and Controls. Data/MC represents measurements and provides events and weights to the Estimators. Physics & Models calculates amplitudes and provides likelihoods to both the Estimators and the Optimizer. The Estimators calculate the discrepancy from the model to the data and provide likelihoods to the Optimizer. The Optimizer varies the model parameters and is connected to the Controls module, which manages the user interface and the run. The background features a cityscape and a 3D surface plot.




Data/MC

represents measurements

- local and global values
- multiple experiments (at once)
- caching

- local and global values
- multiple experiments (at once)
- caching

Physics & Models

calculates amplitudes 

- various formalisms (helicity)
- various models (isobar)
- simple ways to add new modules (wrapper)

- various formalisms (helicity)
- various models (isobar)
- simple ways to add new modules (wrapper)



Controls

user interface &
run manager

amplitude

Estimators

calculates discrepancy from model to data

- function tree
- combined fits and re-fits
- documentation of procedure

calculates discrepancy from
model to data

- function tree
- combined fits and re-fits
- documentation of procedure

Optimizer

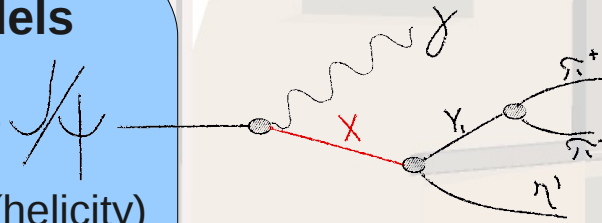
varies model parameter

- interface to external libraries
- various algorithms (gradient decent, genetic, swarm)
- flexible strategies

- interface to external libraries
- various algorithms (gradient decent, genetic, swarm)
- flexible strategies

- local and global values
- multiple experiments (at once)
- caching

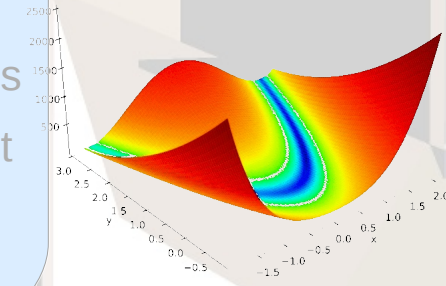
- various formalisms (helicity)
- various models (isobar)
- simple ways to add new modules (wrapper)



user interface &
run manager

- function tree
- combined fits and re-fits
- documentation of procedure

- interface to external libraries
- various algorithms (gradient decent, genetic, swarm)
- flexible strategies



PWA Framework

```
graph TD; DataMC[Data/MC] -- "event weight" --> Estimators; Physics[Physics & Models] -- "amplitude" --> Estimators; Estimators -- "likelihood" --> Optimizer; Optimizer -- "amplitude" --> Physics; Controls[Controls] --- Physics; Controls --- Optimizer;
```

Data/MC

represents measurements

- local and global values
- multiple experiments (at once)
- caching

Physics & Models

calculates amplitudes

- various formalisms (helicity)
- various models (isobar)
- simple ways to add new modules (wrapper)

Estimators

calculates discrepancy from model to data

- function tree
- combined fits and re-fits
- documentation of procedure

Optimizer

varies model parameter

- interface to external libraries
- various algorithms (gradient decent, genetic, swarm)
- flexible strategies

Controls

user interface & run manager

The diagram illustrates the PWA Framework architecture. It consists of five main components: Data/MC, Physics & Models, Estimators, Optimizer, and Controls. Data/MC represents measurements and provides event weights to the Estimators. Physics & Models calculates amplitudes and provides amplitudes to the Estimators. The Estimators calculate the likelihood and provide it to the Optimizer. The Optimizer varies model parameters and provides amplitudes back to the Physics & Models. The Controls module manages the user interface and the run manager, interacting with both the Physics & Models and the Optimizer. The diagram also includes a small tree diagram on the left and a 3D surface plot on the right.

represents measurements

- local and global values
- multiple experiments (at once)
- caching

calculates amplitudes

- various formalisms (helicity)
- various models (isobar)
- simple ways to add new modules (wrapper)

user interface &
run manager

calculates discrepancy from
model to data

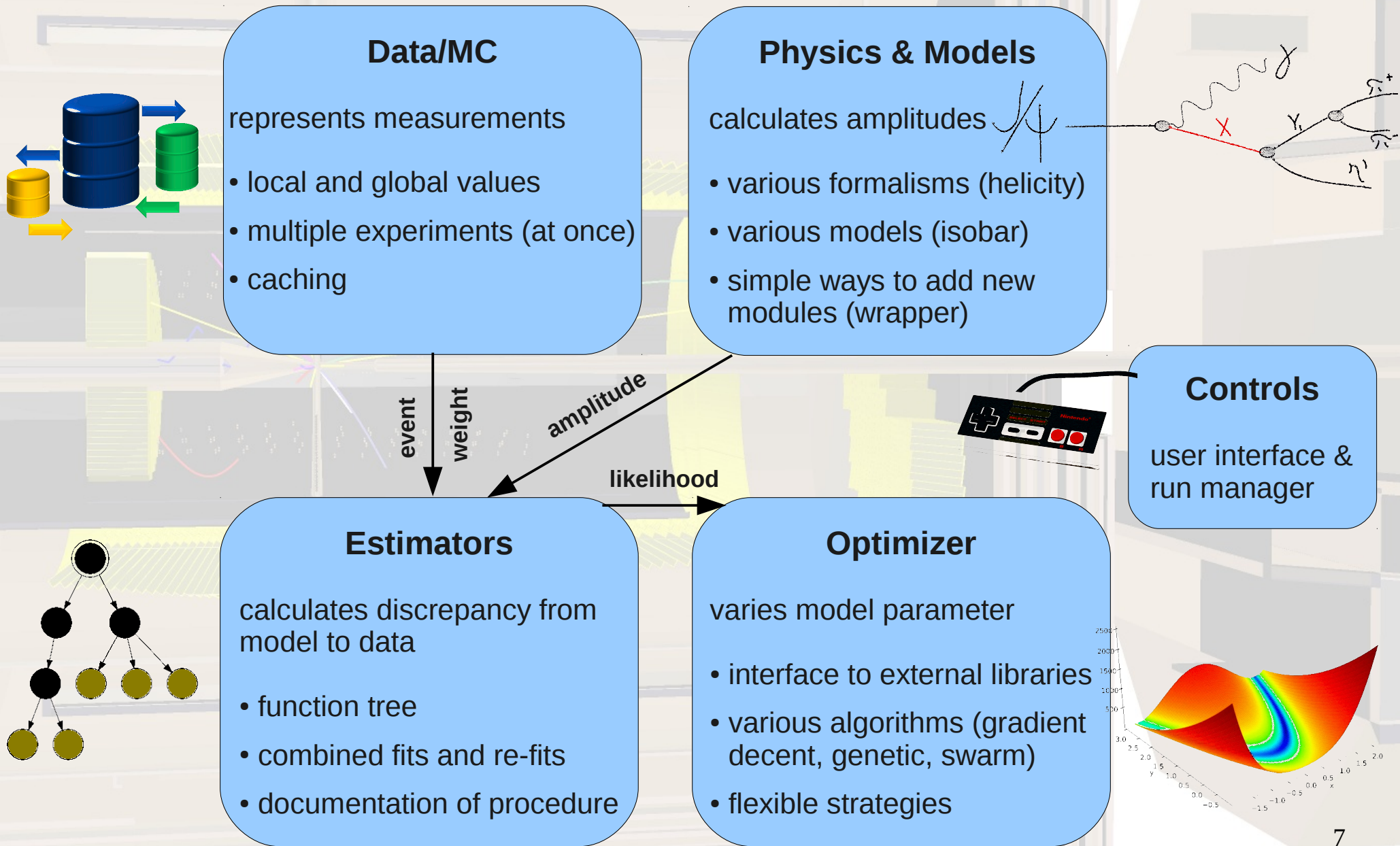
- function tree
- combined fits and re-fits
- documentation of procedure

varies model parameter

- interface to external libraries
- various algorithms (gradient decent, genetic, swarm)
- flexible strategies



PWA Framework



Infrastructure

Build-System: Boost.Build (see talk F. Feldbauer)
Coding, Compiling and Debugging via e.g. Eclipse possible

Version-Control: git (see talk F. Feldbauer)
`git clone gitosis@tau.ep1.rub.de:/var/www/git/ComPWA`

Documentation: Doxygen and DocuWiki

The screenshot shows a web browser displaying the 'ComPWA Wiki' page. The page has a light blue header with the 'PWA Wiki' logo on the left and a 'Logout' button on the right. Below the header is a search bar and links for 'Recent changes', 'Media Manager', and 'Sitemap'. A breadcrumb trail reads 'Trace: • start • git • installation • coding'. The main content area is titled 'ComPWA Coding Conventions' and contains a paragraph about coding conventions, a link to a PDF file, and a section for 'Included Headers' with a list of three items. A 'Table of Contents' sidebar on the right lists various topics like 'Included Headers', 'Classes', 'Namespaces', etc. The page also features a 'Menu' sidebar on the left with links like 'ComPWA DokuWiki', 'About ComPWA', and 'Installation'.

ComPWA Coding Conventions

The following coding conventions should be used when adding code to the framework. This increases readability and makes working with code of different teams easier. The conventions are based on <http://www.ep1.rub.de/~florian/codeConvention.pdf>

Included Headers

1. all system/standard headers in alphabetical order
2. all headers from other packages in alphabetical order
3. all headers from this package in alphabetical order

Classes

- Names of classes should begin with a capital letter
- If a new word starts within the name, this word starts again with a capital letter

Table of Contents

- ♦ ComPWA Coding Conventions
 - ♦ Included Headers
 - ♦ Classes
 - ♦ Namespaces
 - ♦ Forward declarations
 - ♦ Pointers and References
 - ♦ Spaces
 - ♦ Indentation
 - ♦ Line Length
 - ♦ Code Blocks
 - ♦ Functions
 - ♦ Comparison
 - ♦ Documentation

Infrastructure

```

//! Base class for internal parameter.
/*! \class PWAPParameter
 * @file PWAPParameter.hpp
 * This class defines the interface for the internal container of a fit
 * parameter. A parameter consists of a value with optional bounds and error.
 */

```

```

#ifndef _PWAPARAMETER_HPP_
#define _PWAPARAMETER_HPP_

```

```

#include <iostream>
#include <string>
#include <sstream>

```

```

class PWAPParameter
{

```

public:

```

    //! Standard constructor without information
    /*!
     * Standard constructor without information provided.
     */
    PWAPParameter() {
    }

```

Doxygen:

- Docu via comments
- Latex and Html output

```

//! A public function returning a string with parameter information
/*!
 * This function simply returns the member string out_, which contains
 * all parameter information. The string gets rebuild with every change
 * of the parameter.
 * \return string with parameter information
 * \sa operator<<, make_str()
 */
std::string const& to_str() const{
    return out_;
}

```

ComPWA

Common Partial-Wave-Analysis Framework

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Hierarchy	Class Members

PWAPParameter Class Reference

Base class for internal parameter. [More...](#)

#include <PWAPParameter.hpp>

Inheritance diagram for PWAPParameter:

Public Member Functions

	PWAPParameter ()	Standard constructor without information.
virtual	~PWAPParameter ()	Empty Destructor.
virtual const bool	HasBounds () const	Check if parameter has bounds.
virtual const bool	UseBounds () const	Check if bounds should be used.
virtual const bool	HasError () const	Check if parameter has an error.

PWAPParameter
out_ # bounds_ # error_ # usebounds_
+ PWAPParameter() + ~PWAPParameter() + HasBounds() + UseBounds() + HasError()

std::string const& PWAPParameter::to_str () const [inline]

A public function returning a string with parameter information.

This function simply returns the member string out_, which contains all pa

Returns:

string with parameter information

See also:

operator<<, [make_str\(\)](#)

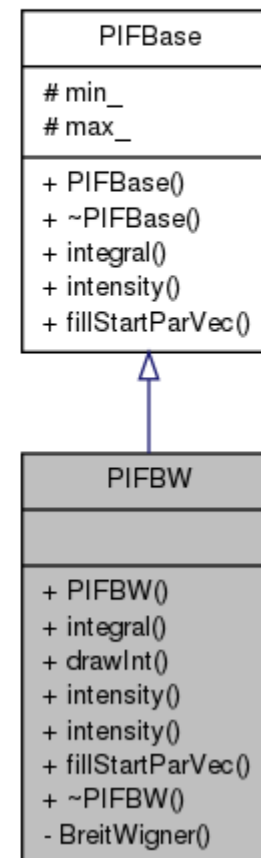
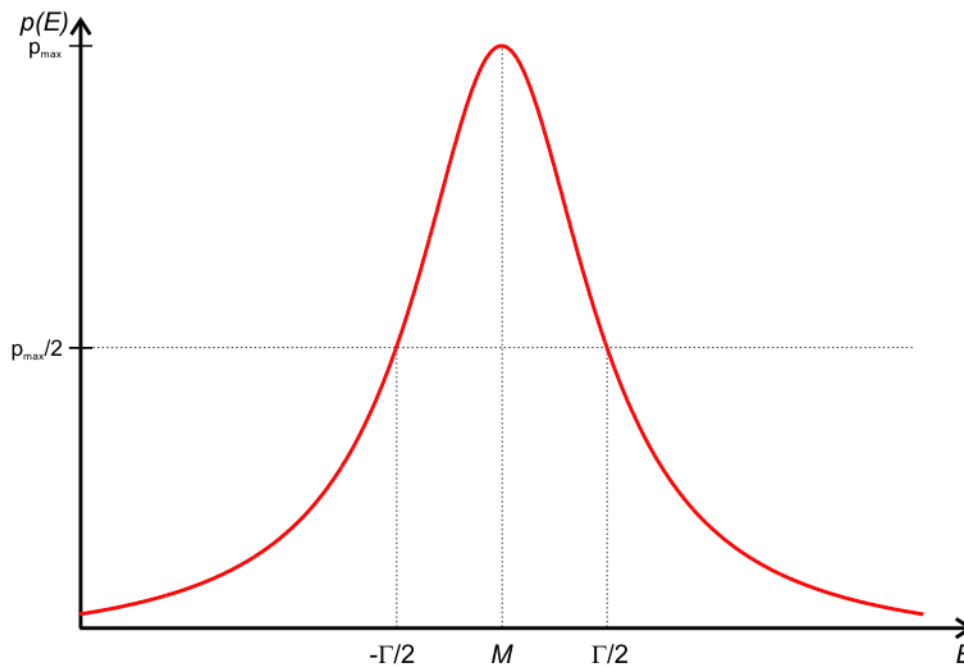
Physics

PIFBW Class Reference

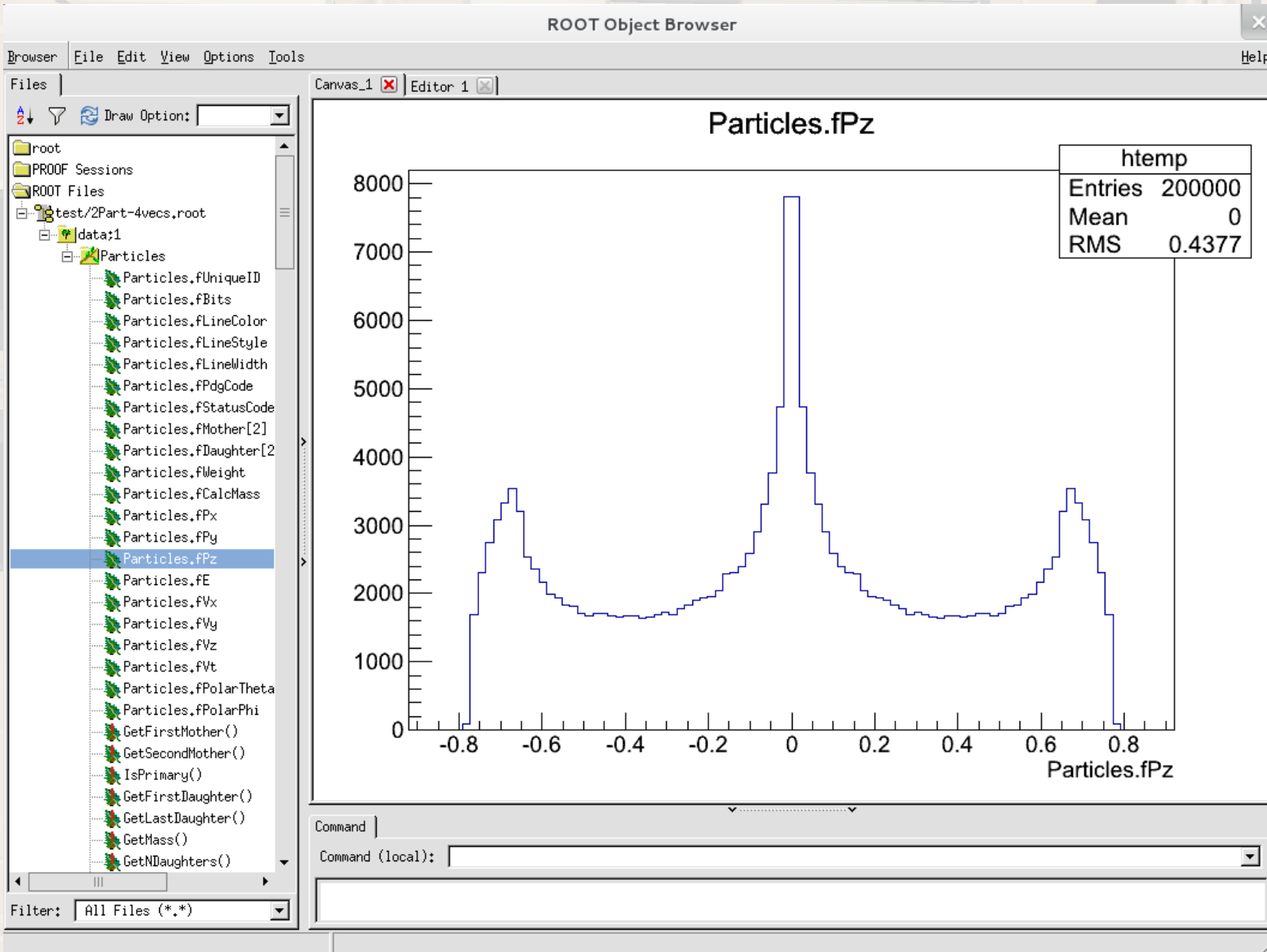
Physics Module with simple 1D Breit-Wigner. [More...](#)

```
#include <PIFBW.hpp>
```

Inheritance diagram for PIFBW:



Side Note: Generator



Hit&Miss
using
TGenPhaseSpace
and PIFBW

2 TParticles per
event stored
in root-file

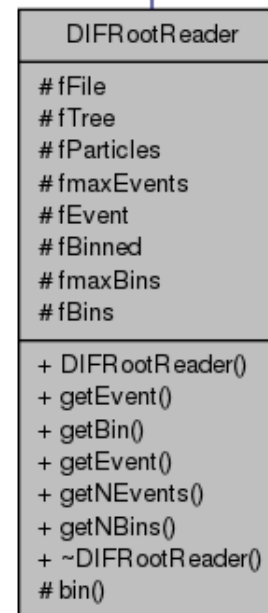
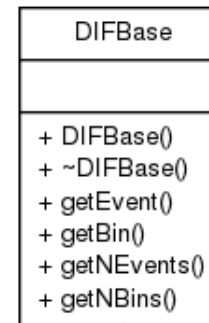
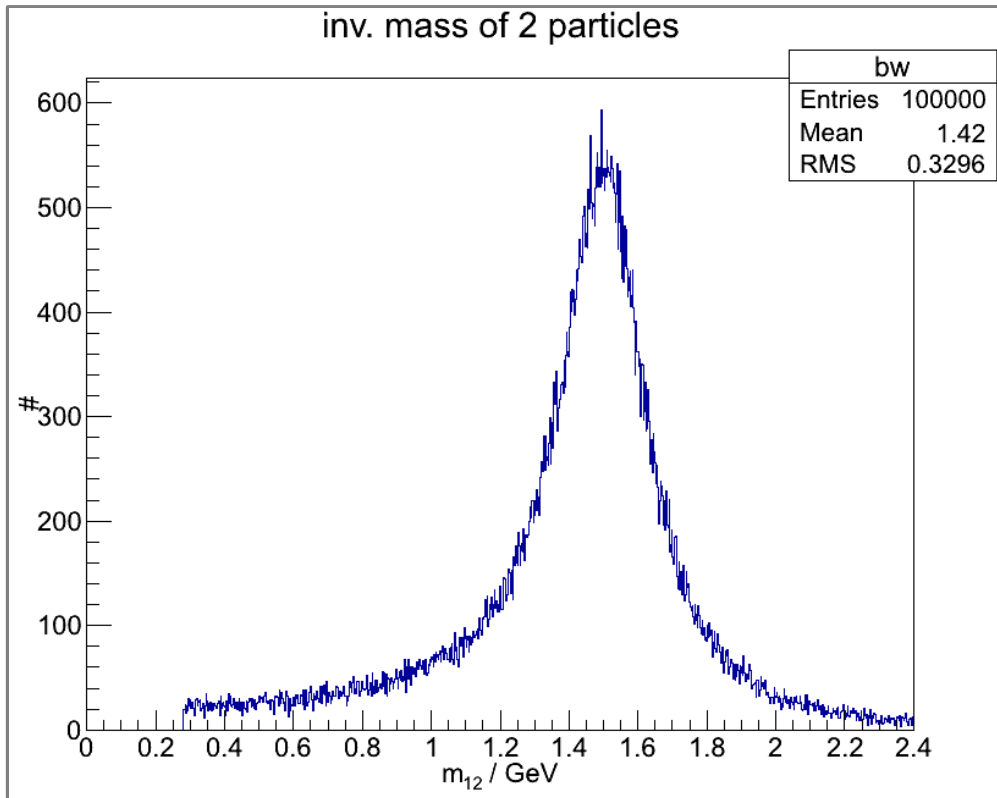
Data

DIFRootReader Class Reference

Reader for data in Root-files. [More...](#)

```
#include <DIFRootReader.hpp>
```

Inheritance diagram for DIFRootReader:



- Reads root-file
- Provides Event Data
- Creates Binned Data

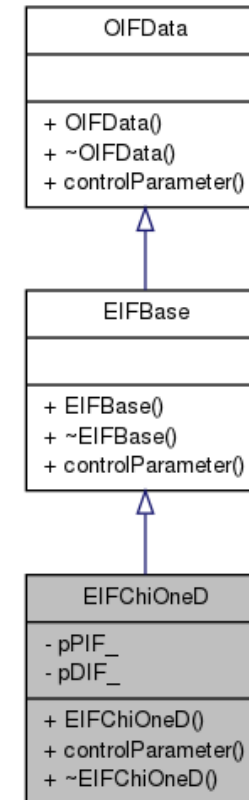
Estimator

EIFChiOneD Class Reference

Simple χ^2 -Estimator. [More...](#)

#include <EIFChiOneD.hpp>

Inheritance diagram for EIFChiOneD:

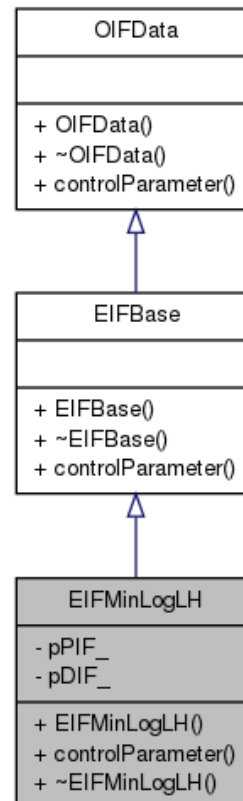


EIFMinLogLH Class Reference

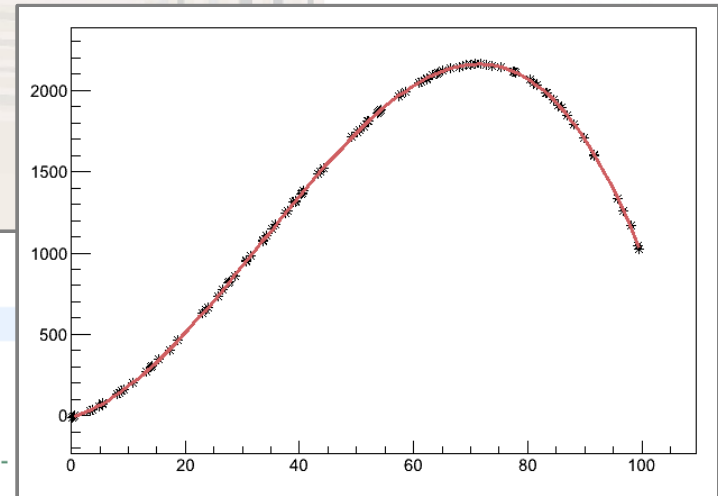
Negative Log Likelihood-Estimator. [More...](#)

#include <EIFMinLogLH.hpp>

Inheritance diagram for EIFMinLogLH:



Optimizer



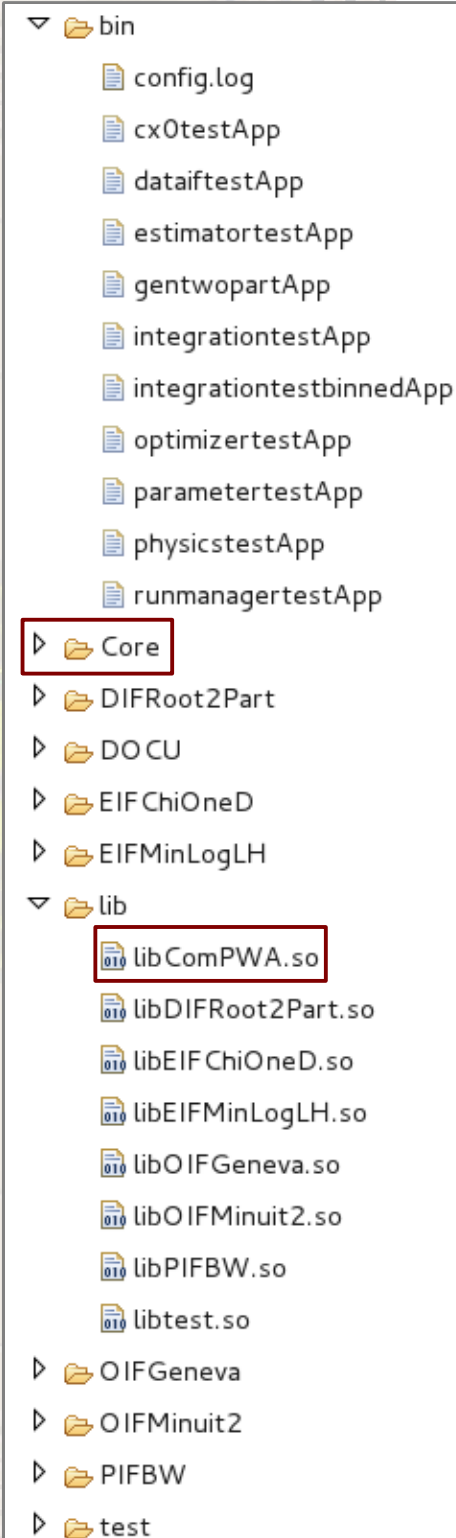
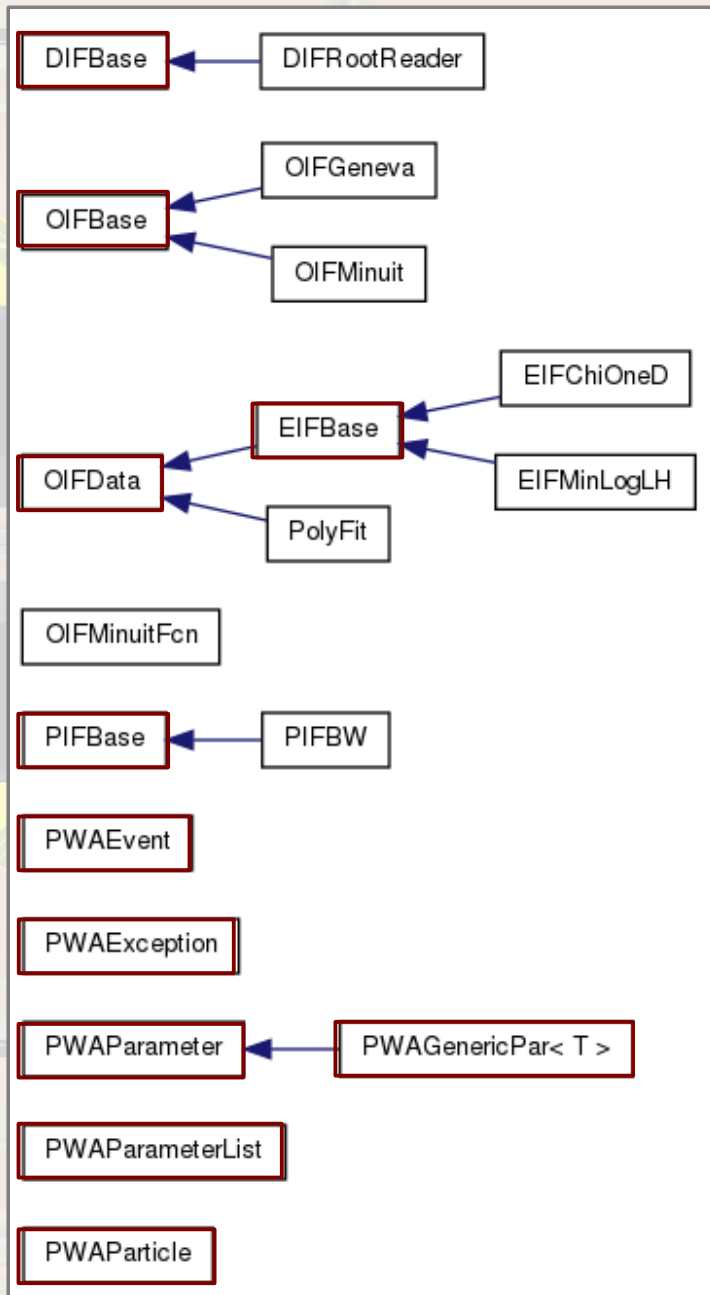
```
int main(int argc, char **argv){
    std::string whichMinimizer("all");
    double p0=-10., p1=10., p2=1., p3=-0.01, sigma_smea=3;
    // Generate data distribution
    std::shared_ptr<PolyFit> myFit(new PolyFit(p0, p1, p2, p3, sigma_smea));

    //-----Minimizer IF -----
    std::vector<std::shared_ptr<OIFBase> > myMinimizerList;
    // Add minimizers
    if (whichMinimizer=="Geneva") myMinimizerList.push_back(std::shared_ptr<OIFBase> (new OIFGeneva(myFit)));
    else if (whichMinimizer=="Minuit") myMinimizerList.push_back(std::shared_ptr<OIFBase> (new OIFMinuit(myFit)));
    else if (whichMinimizer=="all") {
        myMinimizerList.push_back(std::shared_ptr<OIFBase> (new OIFGeneva(myFit)));
        myMinimizerList.push_back(std::shared_ptr<OIFBase> (new OIFMinuit(myFit)));
    }else{
        std::cout << "Minimizer/t" << whichMinimizer << "\tdoesn't exist" << std::endl;
        return 0;
    }

    std::vector<std::shared_ptr<PWAParameter> > par;
    par.push_back(std::shared_ptr<PWAParameter>(new PWAGenericPar<double>(-11,-20,0,3)));
    par.push_back(std::shared_ptr<PWAParameter>(new PWAGenericPar<double>(9.8,5,15,2)));
    par.push_back(std::shared_ptr<PWAParameter>(new PWAGenericPar<double>(1.1,0.5,1.5,0.3)));
    par.push_back(std::shared_ptr<PWAParameter>(new PWAGenericPar<double>(-0.008,-0.02,0,0.005)));
    // Loop over minimizers (at the moment this means: Geneva, Minuit or Geneva then Minuit)
    for(unsigned int Nmin=0; Nmin<myMinimizerList.size(); Nmin++){
        // Pointer to one of the used minimizers
        std::shared_ptr<OIFBase> minimizer = myMinimizerList[Nmin];
        // Do the actual minimization
        double genResult = minimizer->exec(par);

        std::cout << "Minimizer " << Nmin << "\t final par :\t" << genResult << std::endl;
        for(unsigned int i=0; i<par.size(); i++)
            std::cout << "final par " << i << ":\t" << *(par[i]) << std::endl;
        std::cout << "Done ..." << std::endl << std::endl;
    }
}
```

Core



Integration

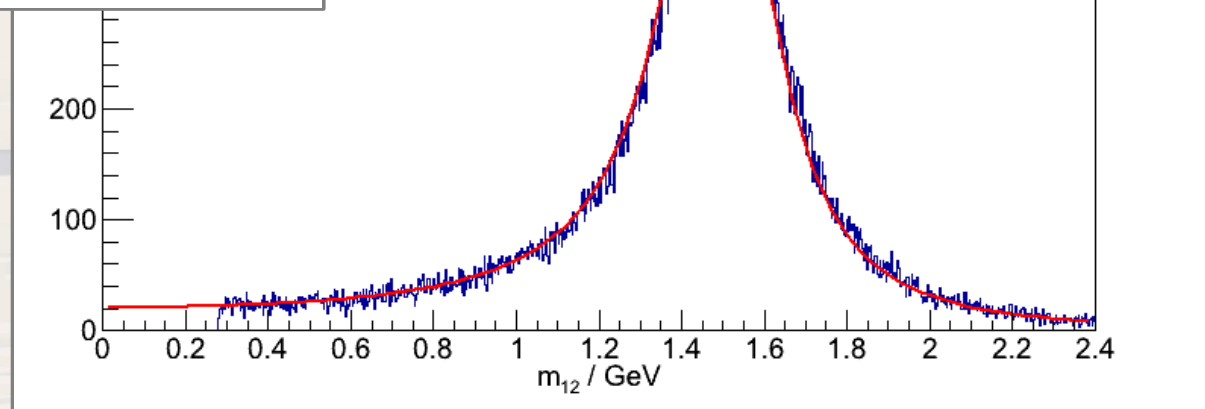
```
int main(int argc, char **argv){
  std::string file="test/2Part-4vecs.root";
  std::cout << "Load Modules" << std::endl;
  std::shared_ptr<DIFBase> myReader(new DIFRootReader(file, false));
  std::shared_ptr<PIFBase> testBW(new PIFBW(0.,5.));
  std::shared_ptr<EIFBase> testEsti(new EIFMinLogLH(testBW, myReader));
  std::shared_ptr<OIFBase> opti(new OIFMinuit(testEsti));

  // Initiate parameters
  std::vector<std::shared_ptr<PWAParameter> > par;
  testBW->fillStartParVec(par);
  par[0]->SetValue(1.7);
  par[1]->SetValue(0.2);

  std::cout << "Initial par :\t" << std::endl;
  std::cout << "initial M:\t" << *(par[0]) << std::endl;
  std::cout << "initial T:\t" << *(par[1]) << std::endl;

  std::cout << "Start Fit" << std::endl;
  double genResult = opti->exec(par);

  std::cout << "Minimized final par :\t" << genResult << std::endl;
  std::cout << "final M:\t" << *(par[0]) << std::endl;
  std::cout << "final T:\t" << *(par[1]) << std::endl;
```



Tests

test/CX0TestApp.cpp	Test-Application to check c++11 support
test/DataFITestApp.cpp	Test-Application of the Root Data-IF
test/EstimatorTestApp.cpp	Test-Application of a simple χ^2 estimator
test/GenTwoPartApp.cpp	Application to generate a two-particle intensity
test/IntegrationTestApp.cpp	Test-Application for full fit with simple modules
test/IntegrationTestBinnedApp.cpp	
test/MinuitTestApp.cpp	Test-Application of the Minuit2 Optimizer-IF
test/OptimizerTestApp.cpp	Test-Application of the Optimizer-IF
test/ParameterTestApp.cpp	Test-Application of internal PWAParameter
test/PhysicsTestApp.cpp	Test-Application of the Physics-IF
test/PolyFit.cpp	
test/PolyFit.hpp [code]	This class derives from OIFData , the data-interface of the optimizers
test/RunManagerTestApp.cpp	Test-Application for fit with simple modules using RunManager

Status

Requirements finished

First standalone test environments a.e.

- Slice-Fit to extract dynamical functions

- 4π generator (for Babar)

- BesIII-analyses

- Pawian

Basic framework set up with

- Module-Interfaces

- Complete trivial 2 particle fit chain:

 - Model for Breit-Wigner

 - Hit & Miss Generator using Model

 - Root-File Reader

 - Binned (Chi2) and unbinned (likelihood) estimators

 - Geneva and Minuit2 optimizer

- Build-System: Boost.Build

- Versioning: git

- Software Documentation: Doxygen & Wiki (not yet public)

ToDo:

- Documentation of fit

- Decay graphs, function trees, models ...

- Control-Module

- Caching, Multi-Threading ...

- Transfer standalone tests to framework

- ...



Thank you!