Optics Studies for the ATLAS Forward Proton Project

Tom Eichlersmith[a]
Advisor: Maciej Trzebinski[b]
[a]Hamline Univseristy, Saint Paul, United States.
[b]Insitute of Nuclear Physics PAN, ul. Radzikowskiego 152, 31-342 Kraków, Poland.

August 10, 2017

1 Introduction

The ATLAS Forward Proton (AFP) detectors [1] are situated about 210m from the Interation Point (IP) of ATLAS along the beam line, so that they are able to measure protons scattered at very small angles (on the order of micro radians) with respect to the beam. There are four AFP detectors, two on each side of the ATLAS IP (Figure 1).

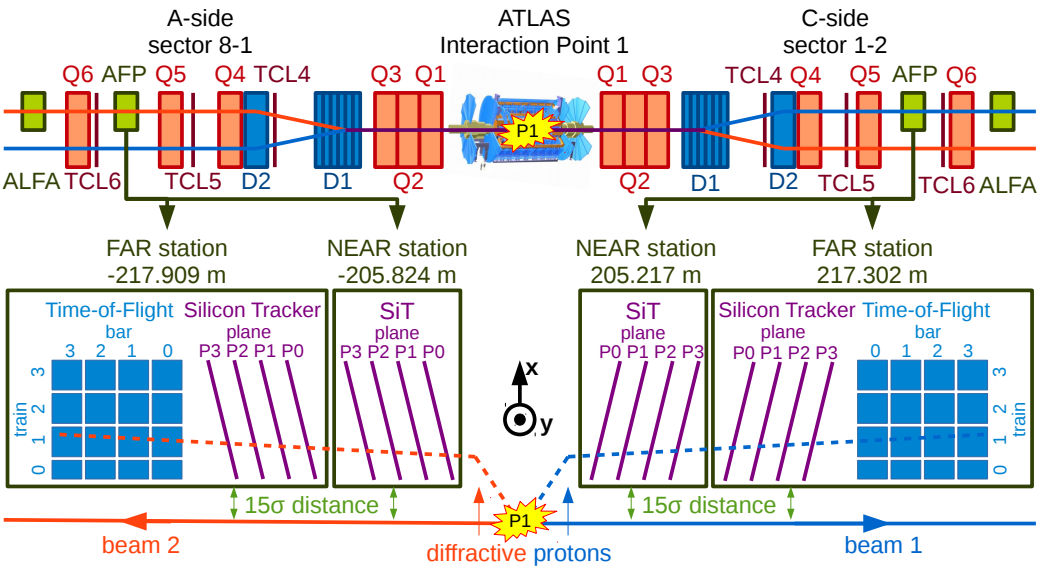


Figure 1: The AFP Scheme [2].

Each detector is capable of measuring the position of protons passing through it. The AFP detectors approach the beam line from the negative x direction in order to detect as many diffracted protons as possible. However, being able to measure the position of the protons at the detectors is not informative unless those measurements can tell something about the kinematics of the protons at the IP. The only way one can make conclusions about IP kinematics from proton positions in AFP detectors is to have a detailed understanding of the trajectories that the protons follow between the IP and the detector. These trajectories are heavily affected by the optic settings of the Large Hadron Collider (LHC) because the protons traverse space controlled by the magnets and collimators used by the LHC to manipulate the beam [1].

There already exists a program (MAD-X [3]) and a package (FPTracker [4]) that are able to perform the foundational job of transforming the information about the optic settings into

a method of simulating the trajectories of protons. MAD-X translates raw information about the LHC optic equipment into so-called twiss files, while FPTracker takes the information from the twiss files in order to simulate the trajectories quickly and efficiently. Before this summer, there were many separate programs that provided the required analysis for this understanding of proton trajectories using MAD-X and FPTracker. This project was about collecting these different programs and putting them into a unified framework that is easier to use and easier to add new methods to. In the next section, the completed methods have been listed and explained, while documentation for the framework (generated by Doxygen) is given in the appendix.

2 The AFPOpticsStudy Framework

The MAD-X program and FPTracker package are still used as the foundational tool for simulating the trajectories of forward protons. More specifically, the MAD-X program is used to generate twiss files according to the specific inputs of the user and FPTracker is used for all of the simulating of forward proton trajectories.

In order to accomplish the goals of easy use and editing, the AFPOpticsStudy framework was built using a object oriented class structure in C++. The class OpticsStudy is the face of the framework, allowing the user to ignore the inner workings of the methods and only be forced to provide the required definitions of optic settings (either through MAD-X files or twiss files) and details of the AFP detectors. However, behind the scenes, different tasks are relegated to separate classes so that users can make changes to parts of the program without affecting other aspects (for example, one can change how a geometric acceptance plot is printed without even looking at the code that generates the acceptance data). This structure also allows the different methods to share the same information, so instead of having to import the optics and detector settings to each method separately, one is able to run all of the methods after importing the optics and detector settings once. This allows the program to be more efficient and less prone to user error. Now each of the incorporated methods will be discussed.

2.1 Geometric Acceptance

Geometric acceptance describes the kinematic limits of protons visible in the detector. The code generates the plot (as shown in Figure 2) using the function OpticsStudy::GeometricAcceptance(). The properties of the plot (including the minimum and maximum transverse momentum, the number of bins, the colors, and the labels) are controlled by the classes GeoAccPars and GeoAccCanvas.

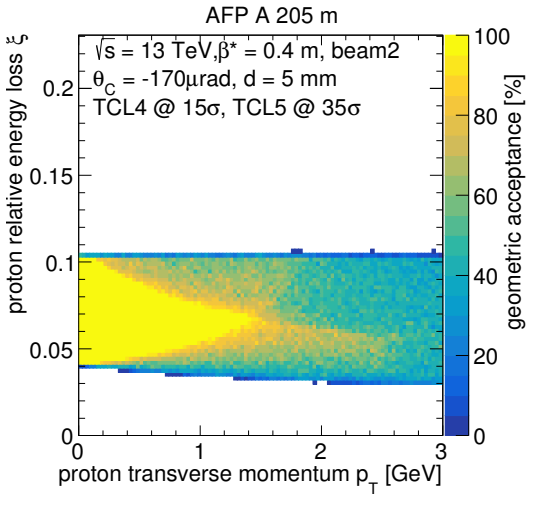
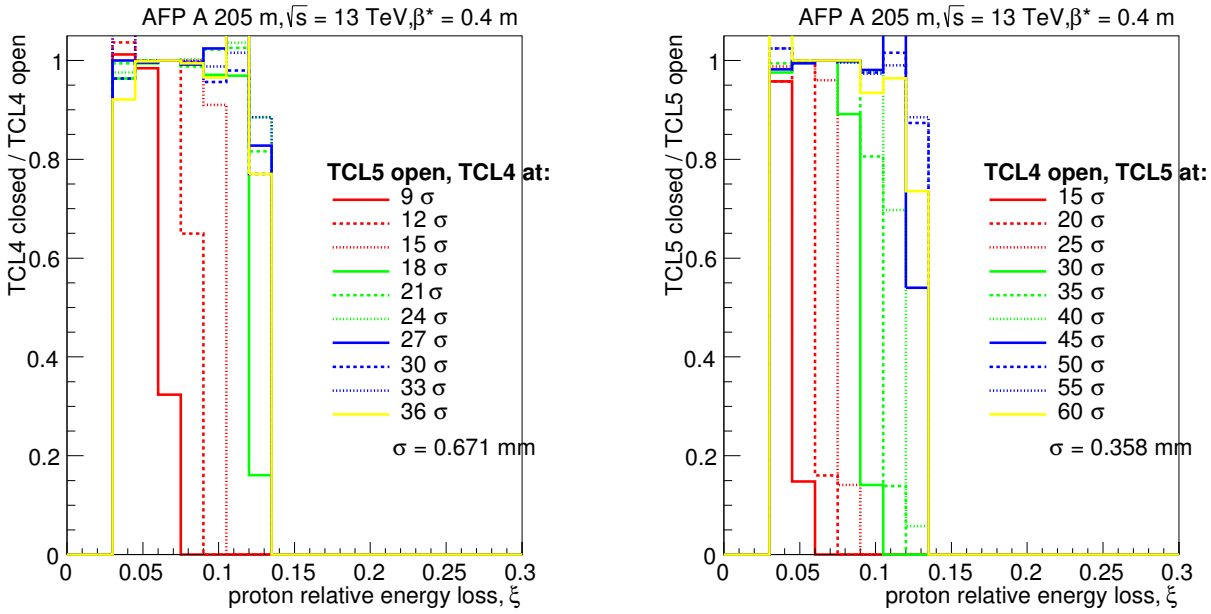


Figure 2: Example geometric acceptance plot.

2.2 Collimator Analysis

The positions of the jaws of the TCL4 and TCL5 collimators have a significant impact on the geometric accpetance of the AFP detectors. In order to study this impact, the collimator analysis method produces four different plots for each detector. Two of them offer information about the effect of the TCL4 and TCL5 collimators individually, while the other two offer information about the combined effect of these two collimators. First, the "closed/open" plots show the ratio between the acceptance at the detector when the collimator is closed a certain amount and the acceptance when the collimator is open (Figure 3). The other plots show the maximum fractional energy loss that a proton can have before the acceptance at the detector decreases to 10% (or 50%) of its value when both collimators are open (Figure 4).



(a) TCL4 (b) TCL5

Figure 3: Open/Closed plots for both collimators.

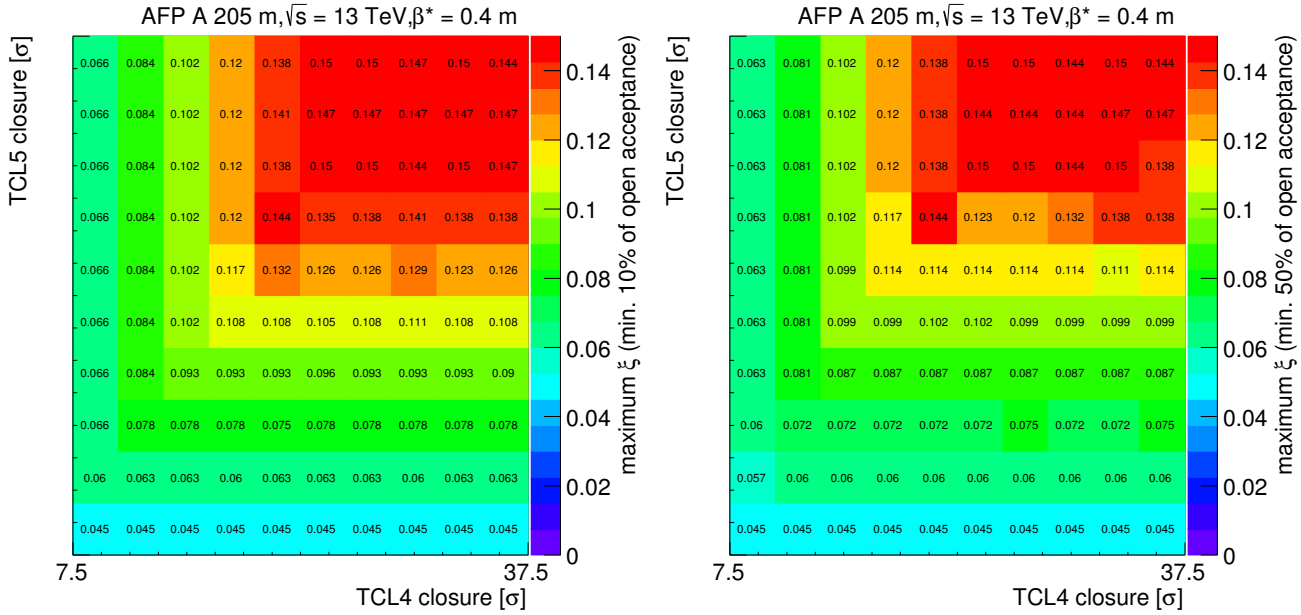


Figure 4: Maximum fractional energy Loss for certain collimator settings.

2.3 Proton Positions

This method produces a plot of the positions of protons in the detector plane. The rectangle on the plot represents the x-y plane of the AFP detector (Figure 5). The plot file names include information about the simulated protons, so multiples of these files can be created in one run of the program.

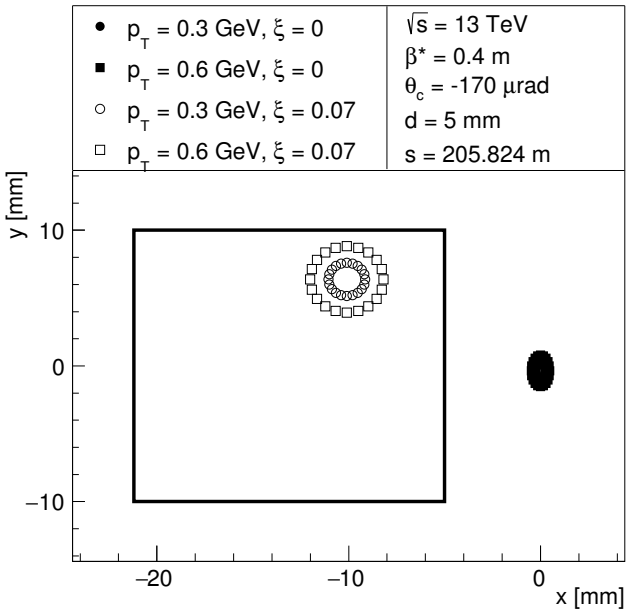


Figure 5: Example proton position at detector plane.

2.4 Example Trajectories

In order to learn about the proton's behaviour while traveling along the beamline, the example trajectory method produces plots of the paths followed by a small set of protons. The plot is created including the equipment that is along the beam between the IP and the AFP detectors. Both x and y directions are ploted and the fractional energy loss and transverse momentum of the protons are varied. The user can choose whether or not to draw the aperture (which is taken directly from the twiss file and the settings of the collimators and AFP detectors).

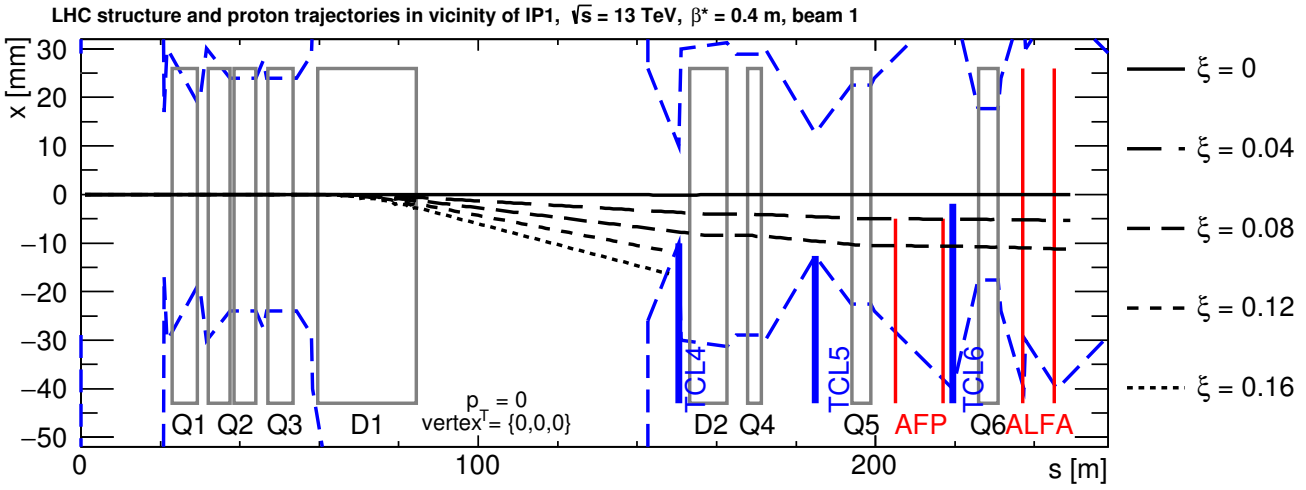


Figure 6: Example trajectories along beam 1.

2.5 Tagged and Timed Probabilities

In order to estimate background reduction needed for physics analysis and feasability studies, this method produces a plot of the probabilities of single tagging and double tagging as a function of pileup at the IP (Figure 7). This simulation is done using the user supplied single tag and double tag probabilities for a single event. The probabilities are depicted in a lin-log and a log-log scale. The user can also input precision of Time-of-Flight (ToF) detectors for the simulation to consider and plot.

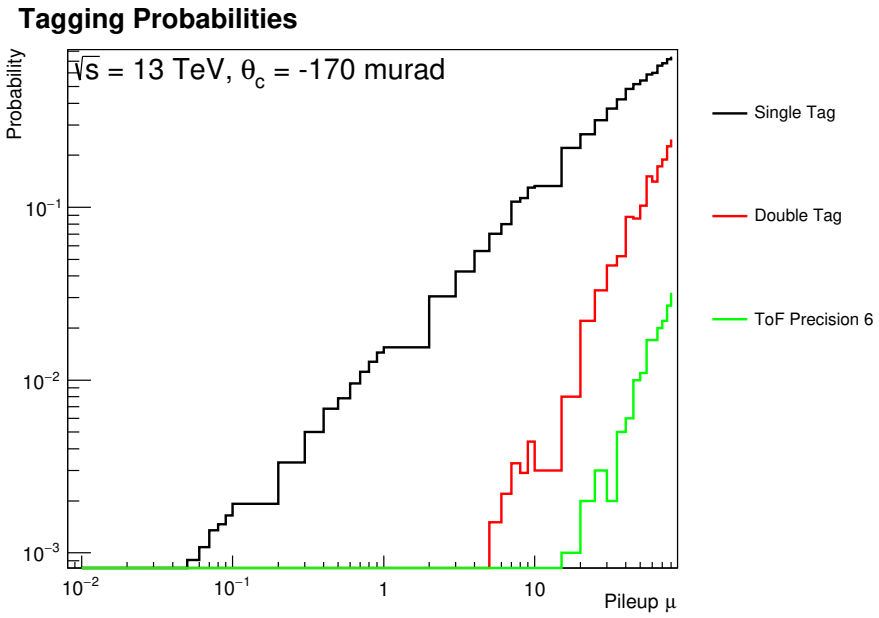


Figure 7: Example tagging probabilities plot.

2.6 Hitting Roman Pot Floor Probabilities

This method produces a plot of the probability of a proton going through the Roman pot floor a certain distance and a plot of the probability of a proton traversing either by or through the Roman pot floor in a certain way (i.e. a "situation" in Figure 8). These plots check the probability of a proton producing a shower. Since a vast majority of protons do not go through the Roman pot floor, this method requires a lot of simulations (and therefore a lot of time) in order to be accurate.

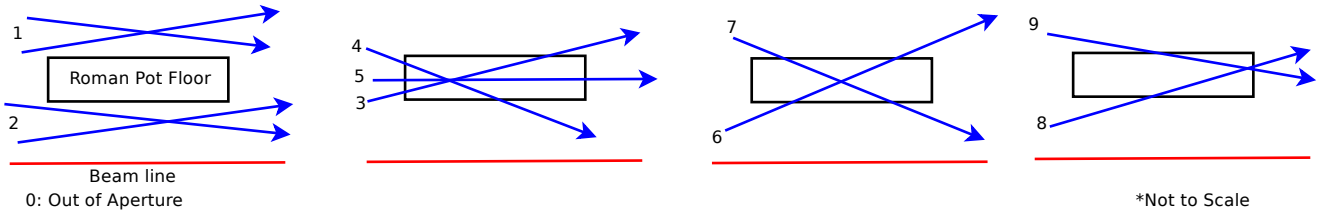
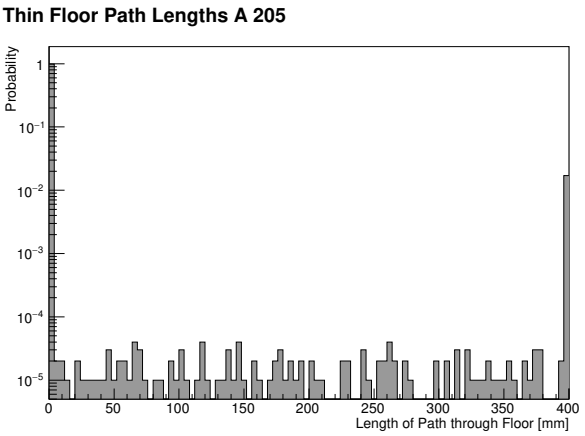


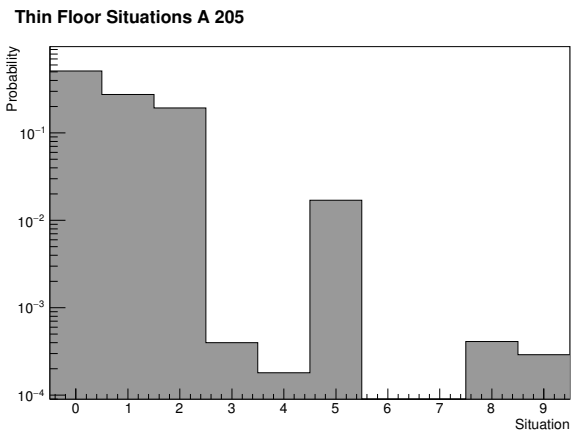
Figure 8: Explanation of number scheme for protons hitting or missing the Roman pot floor.

2.7 Finding and Validating Parameterisation

In order to prepare for kinematic unfolding, the parameterised transport [5] must be obtained. These methods determine the coefficients of the various polynomials in equation (5) from Ap-



(a) Probabilitiy of a certain length.



(b) Probabilitiy of a certain situation.

Figure 9: Thin floor hit probabilities.

pendix A in [1]. These polynomials are allowed to have a range of degrees. Also the validation functions are a simple comparison between the output of a specific parameterisation and the simulation by FPTracker (which the parameterisation was train on). Because the parameterisation methods are more complicated and require more information to be stored, parameterisation classes have been written (Parameterisation, ParametPars, and SpecificParam). These classes allow for editors to work on changing the parameterisation functions (and adding new functions) separately from the other methods that do not use the parameterisation.

2.8 Further Directions

There are some specific methods that were unable to be incorporated into the AFPOpticsStudy framework because of the short duration of this program; however, the foundation of the framework (the classes importing and storing the information from the optic settings) is general enough so that it will be simple to add new methods to the user interface class OpticsStudy. Some additional methods that could be included are more detailed validation methods for the parameterisation and a way to determine the reconstruction resolution of a specific parameterisation. This method could be intertwined with the parameterisation methods so that the program does not have to import a parameterisation from file multiple times. Besides these specific methods, there are undoubtedly more methods that other users will want to add for their own purposes; hopefully this framework allows for new additions to be incorporated easily.

3 Summary

The AFP team will be able to use this code to more efficiently gain information about how the optic settings of the LHC affect the forward protons. This structure has been written in order to isolate the different aspects that each of the separate methods share: the detector settings (AFPStation), the optic settings (OpticsMode), printing (ColAnaCanvas, GeoAccCanvas, etc.), parameter storage (ColAnaPars, GeoAccPars, etc.), and a user interface (OpticsStudy). This isolation allows for easier incorporation of new methods and allows for the current methods to share the same input and output structure, meaning that the user can run the methods at the same time and with the same inputs.

References

[1] L. Adamczyk, et al. *Technical design report for the ATLAS forward proton detector*. No. CERN-LHCC-2015-009. 2015. `cds.cern.ch/record/2017378`

[2] M. Trzebinski, `twiki.cern.ch/twiki/bin/view/Atlas/AFP_Figures`

[3] Schmidt, F., *MAD-X User's Guide*. CERN 2005.

[4] P. Bussey and P. Sherwood, *FPTracker Programme*.

[5] M. Trzebinski, R. Staszewski, and J. Chwastowski, *LHC High Beta* Runs: Transport and Unfolding Methods*, Isrn high energy physics vol. 2012 (2012) 491460. arXiv:1107.2064

AFP Optics Study

Generated by Doxygen 1.8.13

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Namespace Documentation

3.1 AFPOpticsStudy Namespace Reference

Classes

- class AFPStation

  Stores information about a single AFP detector.
- class ColAnaCanvas

  Used to draw and print collimator analysis histograms.
- class ColAnaPars

  Used to store parameters needed for simulating the effect of the collimators.
- class GeoAccCanvas

  Drawing and printing of geometric acceptance plots.
- class GeoAccPars

  Storage of parameters used to obtain geometric acceptance plots.
- class OpticsMode

  Stores values of constants calculated from optics settings.
- class OpticsStudy

  User interface class to run simulations and obtain plots.
- class Parameterisation

  Calculates polynomial values and parameterisation coefficients.
- class ParametPars

  Storage of parameters used in determination of transport parameterisation coefficients.
- class PilPreCanvas

  printing and drawing probabilities that depend on pileup
- class ProPosCanvas

  drawing and printing of example proton positions
- class ProPosPars

  Storage of parameters used in example proton positions at detector plane.
- class Simulator

  Used to perform simulations and generate random numbers.
- class SpecificParam

  Storage and evaluation of parameterisation with specific degrees.
- class TagTimPars

  Storage of parameters used in calculation of tagging and timing probabilities.
- class TrajExCanvas

drawing and printing of example trajectories

- class TrajExPars

  Storage of parameters for simulating example trajectories of the proton.

- class WriteTwiss

  Class Defined to write necessary twiss files using MAD-X program.

Enumerations

- enum DetIndex { **A_FAR**, **A_NEAR**, **C_NEAR**, **C_FAR** }

  Used so that user can decide which detector to focus on.

Variables

- const int **LIN** = 0
- const int **LOG** = 1
- const int **X** = 0
- const int **Y** = 1
- const int **FEL** = 0
- const int **PX** = 1
- const int **PY** = 2
- const DetIndex ALL_DETECTORS [4] = { DetIndex::A_FAR , DetIndex::A_NEAR , DetIndex::C_NEAR , DetIndex::C_FAR }

  namespace constant array for iterating over all four detectors

- const double TwoPI = 6.28318530718

  namespace definition of 2pi

- const double ProtonRestE = 0.938

  namespace definition of the proton rest energy [GeV]

- const double OutOfAperture = 9999.

  namespace definition of a utility constant [m]

3.1.1 Detailed Description

Author

Tom Eichlersmith and Maciej Trzebinski

3.1.2 Enumeration Type Documentation

3.1.2.1 DetIndex

`enum AFPOpticsStudy::DetIndex [strong]`

Used so that user can decide which detector to focus on.

This enum is used solely for convenience, so that the program does not have to periodically check that the detector index is valid.

Chapter 4

Class Documentation

4.1 AFPOpticsStudy::AFPStation Class Reference

Stores information about a single AFP detector.

`#include "AFPOpticsStudy/AFPStation.h"`

Public Member Functions

- AFPStation ()
  Default Constructor.
- AFPStation (double z, double x, double xwidth, double ymax, double safe)
  General Constructor.
- double **Z** () const
- double **X** () const
- double **XWidth** () const
- double **YMax** () const
- double **Safe** () const
- double **FloorDistance** () const
- double **FloorThickness** () const
- double **FloorLength** () const
- double **XPrecision** () const
- double **YPrecision** () const
- double **SXPrecision** () const
- double **SYPrecision** () const
- void **SetZ** (double d)
- void **SetX** (double d)
- void **SetXWidth** (double d)
- void **SetYMax** (double d)
- void **SetSafe** (double d)
- void **SetFloorDistance** (double d)
- void **SetFloorThickness** (double d)
- void **SetFloorLength** (double d)
- void **SetXPrecision** (double p)
- void **SetYPrecision** (double p)
- void **SetSXPrecision** (double p)
- void **SetSYPrecision** (double p)

Private Attributes

- double m_z
  distance from interaction point along beamline in m
- double m_x
  distance from beam center along x-axis in m
- double m_xwidth
  width of detector along x-axis in mm
- double m_ymax
  half-width of detector along y-axis in mm - detector is assumed to be symmetric over y-axis
- double m_safe
  radiation safe distance from detector in m
- double m_floor_d
  distance from roman pot floor to beam in mm
- double m_floor_t
  thickness of thin floor between detector and beam in mm
- double m_floor_l
  length of thin floor between detector and beam in mm
- double m_xpos_precision
  precision of detector in x position in m
- double m_ypos_precision
  precision of detector in y position in m
- double m_xang_precision
  precision of detector in x' in rad
- double m_yang_precision
  precision of detector in y' in rad

4.1.1 Detailed Description

Stores information about a single AFP detector.

In normal operation of this program, all other methods use a pointer to this class, so members of this class can be edited and methods can be rerun with the new information. This class is mainly for convenient storage of the details associated with each of the four AFP detectors, so the only member functions (besides constructors) are accessor functions that help make it easier to pass this stored information from one class to another.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AFPStation() [1/2]

`AFPOpticsStudy::AFPStation::AFPStation ( )`

Default Constructor.

Sets all of the members to defaults: m_z = 210., m_x = 5, m_xwidth = 16.2, m_ymax = 10, m_safe = 1, m_floor_d = m_x-0.5, m_foor_t = 0.4, m_floor_l = 400, m_xpos_precision = 1e-7, m_ypos_precision = 1e-7, m_xang_precision = 1e-8, m_yang_precision = 1e-8

4.1.2.2 AFPStation() [2/2]

```
AFPOpticsStudy::AFPStation::AFPStation (
            double z,
            double x,
            double xwidth,
            double ymax,
            double safe )
```

General Constructor.

Sets all of the corresponding member variables to inputs and all others to calculated defaults (see AFPStation())

The documentation for this class was generated from the following files:

- AFPOpticsStudy/AFPOpticsStudy/AFPStation.h
- AFPOpticsStudy/src/AFPStation.cpp

4.2 AFPOpticsStudy::ColAnaCanvas Class Reference

Used to draw and print collimator analysis histograms.

```
#include <ColAnaCanvas.h>
```

Public Member Functions

- ColAnaCanvas ()
  Default Constructor.
- ColAnaCanvas (const OpticsMode *mode, const AFPStation *afp, const ColAnaPars *cap)
  Basic Constructor.
- ~ColAnaCanvas ()
  Destructor.
- void DrawMaxfEl (TH2F *hist, std::string min_acceptance)
  Draws histogram corresponding to the maximum fractional energy loss allowed for pairs of collimator settings.
- void DrawClosedOpen (std::vector< TH1F *> hists, std::vector< int > settings, bool tcl4_open)
  Draws the list of histograms corresponding to different settings for collimators.
- void Save (std::string filepath)
  Saves whatever is on the canvas to the input filepath.

Private Member Functions

- void SetROOTStyle ()
  Sets global printing and drawing style.
- void BuildLabels ()
  Builds objects used to label the plot from information stored in member variables.
- void DeleteLabels ()
  Deletes objects used to label the plot.

Private Attributes

- const OpticsMode * m_mode
  current mode
- const ColAnaPars * m_cap
  simulation parameters
- const AFPStation * m_afp
  current afp station
- TCanvas * m_colanacanvas
  canvas used to draw on
- TLatex * m_beaminfo
  object storing info about the beam
- TLegend * m_legend_co [2]
  legends corresponding to different collimators

4.2.1 Detailed Description

Used to draw and print collimator analysis histograms.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 ColAnaCanvas() [1/2]

```
AFPOpticsStudy::ColAnaCanvas::ColAnaCanvas ( )
```

Default Constructor.

Sets global printing and drawing style and intializes the canvas

See also

SetROOTStyle()

4.2.2.2 ColAnaCanvas() [2/2]

```
AFPOpticsStudy::ColAnaCanvas::ColAnaCanvas (
            const OpticsMode * mode,
            const AFPStation * afp,
            const ColAnaPars * cap )
```

Basic Constructor.

Sets member variables to corresponding inputs, sets global printing and drawing style, initializes the canvas, and builds the plot labels

See also

SetROOTStyle(), BuildLabels()

4.2.2.3 ~ColAnaCanvas()

```
AFPOpticsStudy::ColAnaCanvas::~ColAnaCanvas ( )
```

Destructor.

See also

DeleteLabels()

4.2.3 Member Function Documentation

4.2.3.1 DrawClosedOpen()

```
void AFPOpticsStudy::ColAnaCanvas::DrawClosedOpen (
            std::vector< TH1F *> hists,
            std::vector< int > settings,
            bool tcl4_open )
```

Draws the list of histograms corresponding to different settings for collimators.

Assumes histograms are not styled. Draws all histograms in hists onto the same canvas.

Parameters

hists	vector of histograms storing acceptance compared to acceptance for the open setting
settings	vector that stores the corresponding setting of collimator for each histogram in hists
tcl4_open	defining if TCL4 is set to open (true) or if TCL5 is set to open (false)

4.2.3.2 DrawMaxfEl()

```
void AFPOpticsStudy::ColAnaCanvas::DrawMaxfEl (
            TH2F * hist,
            std::string min_acceptance )
```

Draws histogram corresponding to the maximum fractional energy loss allowed for pairs of collimator settings.

Assumes histogram is not styled at all.

Parameters

hist	two-dimensional histogram storing the maximum fractional energy loss allowed for each pair of settings
min_acceptance	the minimum acceptance (%) used to calculate the maximum fractional energy loss

The documentation for this class was generated from the following files:

- AFPOpticsStudy/AFPOpticsStudy/canvases/ColAnaCanvas.h
- AFPOpticsStudy/src/canvases/ColAnaCanvas.cpp

4.3 AFPOpticsStudy::ColAnaPars Class Reference

Used to store parameters needed for simulating the effect of the collimators.

`#include <ColAnaPars.h>`

Public Member Functions

- ColAnaPars ()
  Default Constructor.
- ColAnaPars (double fElMAX, int nevents, int cbins, int zbins, int rebin)
  Basic Constructor.
- int c1set (int i) const
  Calculates corresponding collimator width setting for the given index i.
- int c2set (int i) const
  Calculates corresponding collimator width setting for the given index i.
- double **FracELossMAX** () const
- double **FracELossSTEP** () const
- int **NEvents** () const
- int **NColSettings** () const
- int **NFracELossBins** () const
- int **Rebin** () const
- void **SetFracELossMAX** (double f)
- void **SetNEvents** (int n)
- void **SetNColSettings** (int n)
- void **SetNFracELossBins** (int n)
- void **SetRebin** (int n)

Private Member Functions

- void CheckRebin ()
  Checks if m_rebin is a divisor of m_zbins (if not, sets m_rebin to 1)
- void SetfElSTEP ()
  Sets m_fElSTEP = m_fElMAX/m_zbins.

Private Attributes

- double m_fElMAX
  Maximum fractional energy loss to simulate.
- double m_fElSTEP
  Difference between fractional energy loss bins (set to m_fElMAX/m_zbins)
- int m_num_events
  Number of events to simulate for each fractional energy loss bin.
- int m_cbins
  Number of different collimator settings.
- int m_zbins
  Number of different fractional energy loss bins.
- int m_rebin
  Number of bins to combine when rebinning for the open/closed plots (Must be a divisor of m_zbins)

4.3.1 Detailed Description

Used to store parameters needed for simulating the effect of the collimators.

While this class is mainly used to conveniently pass the simulation parameters of collimator analysis between other classes, it also contains member functions that make sure the different parameters are set correctly and it contains the functions that translate the indices of a storage array to the actual integer setting of the collimator widths (c1set() and c2set).

4.3.2 Constructor & Destructor Documentation

4.3.2.1 ColAnaPars() [1/2]

```
AFPOpticsStudy::ColAnaPars::ColAnaPars ( )  [inline]
```

Default Constructor.

Sets member variables to default settings: Maximum fractional energy loss = 0.3, Number of Events = 100, Number of Collimator Settings = 11, Number of Fractional Energy Loss Bins = 100, Rebin Number = 5

4.3.2.2 ColAnaPars() [2/2]

```
AFPOpticsStudy::ColAnaPars::ColAnaPars (
            double fElMAX,
            int nevents,
            int cbins,
            int zbins,
            int rebin )  [inline]
```

Basic Constructor.

Sets member variables to inputs

Parameters

fElMAX	maximum fractional energy loss to simulate
nevents	number of events to simulate fore each fractional energy loss bin
cbins	number of collimator settings
zbins	number of fractional energy loss bins
rebin	number of bins to combine when making open/closed plots

The documentation for this class was generated from the following file:

- AFPOpticsStudy/AFPOpticsStudy/simulationparameters/ColAnaPars.h

4.4 AFPOpticsStudy::GeoAccCanvas Class Reference

Drawing and printing of geometric acceptance plots.

`#include <GeoAccCanvas.h>`

Public Member Functions

- GeoAccCanvas ()
  Default Constructor.
- GeoAccCanvas (const OpticsMode *mode, const AFPStation *afp, const GeoAccPars *gap)
  Basic Constructor.
- ~GeoAccCanvas ()
  Destructor.
- void Draw (TH2D &hist)
  Draws histogram as a geometric acceptance plot.
- void Save (std::string filepath)
  Saves current canvas to filepath.

Private Member Functions

- void SetROOTStyle ()
  Sets global drawing and printing style.
- void BuildLabels ()
  Builds plot labels using current mode, afp station, and parameters.
- void DeleteLabels ()
  Deletes plot labels.

Private Attributes

- const OpticsMode * m_mode
  current mode
- const AFPStation * m_afp
  current afp station
- const GeoAccPars * m_gap
  geometric acceptance parameters
- TCanvas * m_geoacccanvas
  canvas for drawing and printing
- TLatex * m_title
  plot label to make detailed title
- TLatex * m_beaminfo1
  plot label storing beam details
- TLatex * m_beaminfo2
  plot label storing beam details
- TLatex * m_colinfo
  plot label storing collimator details

4.4.1 Detailed Description

Drawing and printing of geometric acceptance plots.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 GeoAccCanvas() [1/2]

```
AFPOpticsStudy::GeoAccCanvas::GeoAccCanvas ( )
```

Default Constructor.

Sets global style and intializes canvas

See also

SetROOTStyle()

4.4.2.2 GeoAccCanvas() [2/2]

```
AFPOpticsStudy::GeoAccCanvas::GeoAccCanvas (
            const OpticsMode * mode,
            const AFPStation * afp,
            const GeoAccPars * gap )
```

Basic Constructor.

Sets member variables to corresponding inputs, sets global style, initializes canvas, and builds plot labels

See also

SetROOTStlye(), BuildLabels()

4.4.2.3 ~GeoAccCanvas()

```
AFPOpticsStudy::GeoAccCanvas::~GeoAccCanvas ( )
```

Destructor.

See also

DeleteLabels()

4.4.3 Member Function Documentation

4.4.3.1 Draw()

```
void AFPOpticsStudy::GeoAccCanvas::Draw (
            TH2D & hist )
```

Draws histogram as a geometric acceptance plot.

Assumes input histogram is not styled at all. Draws labels on top of histogram. Assumes the x-axis is transverse momentum and the y axis is fractional energy loss.

Parameters

hist	histogram storing geometric acceptance information

The documentation for this class was generated from the following files:

- AFPOpticsStudy/AFPOpticsStudy/canvases/GeoAccCanvas.h
- AFPOpticsStudy/src/canvases/GeoAccCanvas.cpp

4.5 AFPOpticsStudy::GeoAccPars Class Reference

Storage of parameters used to obtain geometric acceptance plots.

`#include <GeoAccPars.h>`

Public Member Functions

- GeoAccPars ()
  Default Constructor.
- GeoAccPars (double E_min, double pT_min, double pT_max, int nbins_E, int nbins_pT, int ntrays)
  Basic Constructor.
- double **MinimumE** () const
- double **MinimumPT** () const
- double **MaximumPT** () const
- int **NBinsE** () const
- int **NBinsPT** () const
- int **NTrialsPerBin** () const
- void **SetMinimumE** (double e)
- void **SetMinimumPT** (double p)
- void **SetMaximumPT** (double p)
- void **SetNBinsE** (int n)
- void **SetNBinsPT** (int n)
- void **SetNTrialsPerBin** (int n)

Private Attributes

- double m_E_min
  [GeV] Minimum energy to consider in geometric acceptance (maximum E is beam energy)
- double m_pT_min
  [GeV] minimum transverse momentum to consider in geometric acceptance
- double m_pT_max
  [GeV] maximum transverse momentum to consider in geometric acceptance
- int m_nbins_E
  Number of bins to use in geometric acceptance for energy.
- int m_nbins_pT
  Number of bins to use in geometric acceptance for transverse momentum.
- int m_ntrays
  Number of trials for each specific bin in geometric acceptance.

4.5.1 Detailed Description

Storage of parameters used to obtain geometric acceptance plots.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 GeoAccPars() [1/2]

```
AFPOpticsStudy::GeoAccPars::GeoAccPars ( )  [inline]
```

Default Constructor.

Sets member variables to default values: Minimum Energy = 5000, Transverse Momentum Range: [0.,3.0], Energy Bins = 100, Transverse Momentum Bins = 100, Trials per bin = 100,

4.5.2.2 GeoAccPars() [2/2]

```
AFPOpticsStudy::GeoAccPars::GeoAccPars (
            double E_min,
            double pT_min,
            double pT_max,
            int nbins_E,
            int nbins_pT,
            int ntrays )  [inline]
```

Basic Constructor.

Sets member variables to corresponding inputs

The documentation for this class was generated from the following file:

- AFPOpticsStudy/AFPOpticsStudy/simulationparameters/GeoAccPars.h

4.6 AFPOpticsStudy::OpticsMode Class Reference

Stores values of constants calculated from optics settings.

```
#include "AFPOpticsStudy/OpticsMode.h"
```

Public Member Functions

- OpticsMode ()

  Default Constructor.
- OpticsMode (std::string twissdir, bool isb2, double ca, double normex, double normey, double stprob, double dtprob)

  Default Set Collimator Constructor.
- OpticsMode (std::string twissdir, bool isb2, double ca, double normex, double normey, double stprob, double dtprob, int tcl4, int tcl5)

  Sigma Set Collimator Constructor.
- OpticsMode (std::string twissdir, bool isb2, double ca, double normex, double normey, double stprob, double dtprob, double tcl4, double tcl5)

  Meter Set Collimator Constructor.
- void List () const

  Prints values of all member variables to std::cout (except vectors storing the aperture)
- double **MinAperAcc** () const
- double **BeamEnergy** () const
- int **SqrtSTeV** () const
- double **SpreadPz** () const
- double **BunchLength** () const
- double **CrossingAngle** () const
- double **SingleTagProb** () const
- double **DoubleTagProb** () const
- double **BetaX_IP** () const
- double **BetaY_IP** () const
- double **BeamXSpread** () const
- double **BeamYSpread** () const
- double **SpreadPx** () const
- double **SpreadPy** () const
- double **SigmaXTCL4** () const
- double **SigmaXTCL5** () const
- int **TCL4sig** () const
- int **TCL5sig** () const
- double **TCL4m** () const
- double **TCL5m** () const
- double **SigmaXNearAFP** () const
- double **SigmaXFarAFP** () const
- std::vector< double > **ApertureX** () const
- std::vector< double > **ApertureY** () const
- std::vector< double > **ApertureS** () const
- std::string **TwissDirPath** () const
- bool **Beam** () const
- int **BeamNum** () const
- char **Arm** () const
- void **SetSingleTagProb** (double st)
- void **SetDoubleTagProb** (double dt)
- void **SetTCL4sig** (int n)
- void **SetTCL5sig** (int n)
- void **SetTCL4m** (double w)
- void **SetTCL5m** (double w)
- void **SetMinAperAcc** (double w)

Private Member Functions

- int ElementParameterPosition (std::string line, std::string parameter)

  Obtains column numbers for specific parameter.
- std::string OpticsParameter_string (std::string parameter)

  Obtains the OpticsParameter parameter from the twiss file.
- double OpticsParameter_double (std::string parameter)

  Obtains the OpticsParameter parameter from the twiss file.
- double ElementParameter (std::string element, std::string parameter)

  Obtains the Element Parameter parameter from the twiss file.
- void ReadTwissFile ()

  Reads twiss file and stores requred values in members.
- void ReadApertureWidths ()

  Reads aperture widths and corresponding s-positions.
- void Initialize (std::string twissdir, bool isb2, double crossing_angle, double normex, double normey, double stprob, double dtprob)

  SetUp class using inputs.
- void ListVal (std::string var_name, double var) const

  prints varname to std::cout
- void ListVal (std::string var_name, int var) const

  prints varname to std::cout

Private Attributes

- std::string m_sequence

  Input sequence for this mode.
- double m_beam_energy

  Energy of the Beam [GeV].
- double m_pc

  PC value read from twiss file [GeV].
- double m_gamma_rel

  Relativistic gamma read from twiss file.
- double m_beta_rel

  Relativistic beta read from twiss file.
- double m_spread_pz

  Spread in z-momentum, approximated by multiplying spread in energy by m_pc [GeV].
- double m_bunch_length

  Length of bunches[m].
- double m_geometric_emittance_x

  Geometric Emittance read from twiss file [m∗rad].
- double m_geometric_emittance_y

  Geometric Emittance read from twiss file [m∗rad].
- double m_crossing_angle

  Crossing angle as input by user [murad].
- double m_norm_emittance_x

  Normalized Emittance [m∗rad].
- double m_norm_emittance_y

  Normalized Emittance [m∗rad].
- double m_single_tag_prob

  Probability of a single proton being tagged input by user.

- double m_double_tag_prob

  Probability of protons on both sides being tagged input by user.
- double m_s_IP

  s-position of IP read from twiss file (should be zero if twiss file was generated correctly) [m]
- double m_beta_x_IP

  beta function read from twiss file [m]
- double m_beta_y_IP

  beta function read from twiss file [m]
- double m_gamma_x_IP

  Courant-Snyder gamma read from twiss file.
- double m_gamma_y_IP

  Courant-Snyder gamma read from twiss file.
- double m_beam_x_spread

  Calculated using beta function, emittance and relativistic constants [m].
- double m_beam_y_spread

  Calculated using beta function, emittance and relativistic constants [m].
- double m_spread_px

  Calculated from beam angular spread, emittance and relativistic constants [GeV].
- double m_spread_py

  Calculated from beam angular spread, emittance and relativistic constants [GeV].
- double m_sigmax_tcl4

  width of beam at TCL4 collimator (calculated from twiss file constants) [m]
- double m_sigmax_tcl5

  width of beam at TCL5 collimator (calculated from twiss file constants) [m]
- int m_n_tcl4

  Integer multiplying beam width that defines setting of TCL4 (set by user) [sigma].
- int m_n_tcl5

  Integer multiplying beam width that defines setting of TCL5 (set by user) [sigma].
- double m_widthx_tcl4

  Width setting of TCL4 (calculated from beam width and integer setting or directly input) [m].
- double m_widthx_tcl5

  Width setting of TCL5 (calculated from beam width and integer setting or directly input) [m].
- double m_sigmax_near_afp

  Beam width at afp station nearest to IP [m].
- double m_sigmax_far_afp

  Beam width at afp station further from IP [m].
- std::vector< double > m_aperture_x

  Values for aperture size in the x direction (read from twiss file) [m].
- std::vector< double > m_aperture_y

  Values for aperture size in the y direction (read from twiss file) [m].
- std::vector< double > m_aperture_s

  Values of s-displacement corresponding to the aperture sizes (read from twiss file) [m].
- double m_min_aperture_accepted

  Minimum value of aperture size accepted (set to 0.001 by default) [m].
- std::string m_twiss_dir_path

  Path to directory containing twiss files.
- std::string m_twiss_file_path

  Path to twiss file that is read into this class.
- bool m_beam

  Definition of beam (false <=> beam1 , true <=> beam2)
- char m_arm

  Definition of beam (beam1 <=> C , beam2 <=> A)

4.6.1 Detailed Description

Stores values of constants calculated from optics settings.

Reads in most values directly from given twiss files. One instance of this class corresponds to one beam (either 1 or 2) and one optics settings. Similar to AFPStation, this class is mainly used for convenient storage of the details associated with each beam. This means that the only functions that are not accessors are ones used to read in the values of the parameters from the input twiss directory path. Some additional accessor functions are defined (like BeamNum() and SqrtSTev()) for convenience.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 OpticsMode() [1/4]

```
AFPOpticsStudy::OpticsMode::OpticsMode ( )
```

Default Constructor.

Leaves class blank

4.6.2.2 OpticsMode() [2/4]

```
AFPOpticsStudy::OpticsMode::OpticsMode (
            std::string twissdir,
            bool isb2,
            double ca,
            double normex,
            double normey,
            double stprob,
            double dtprob )
```

Default Set Collimator Constructor.

Initializes class and sets collimator widths to defaults

See also

Initialize(), SetTCL4sig(), SetTCL5sig(), ReadApertureWidths()

4.6.2.3 OpticsMode() [3/4]

```
AFPOpticsStudy::OpticsMode::OpticsMode (
            std::string twissdir,
            bool isb2,
            double ca,
            double normex,
            double normey,
            double stprob,
            double dtprob,
            int tcl4,
            int tcl5 )
```

Sigma Set Collimator Constructor.

Initializes class and sets collimator widths in terms of a multiple of the beam width

See also

Initialize(), SetTCL4sig(), SetTCL5sig(), ReadApertureWidths()

4.6.2.4 OpticsMode() [4/4]

```
AFPOpticsStudy::OpticsMode::OpticsMode (
            std::string twissdir,
            bool isb2,
            double ca,
            double normex,
            double normey,
            double stprob,
            double dtprob,
            double tcl4,
            double tcl5 )
```

Meter Set Collimator Constructor.

Initializes class and sets collimator widths in meters

See also

Initialize(), SetTCL4m(), SetTCL5m(), ReadApertureWidths()

4.6.3 Member Function Documentation

4.6.3.1 ElementParameter()

```
double AFPOpticsStudy::OpticsMode::ElementParameter (
            std::string element,
            std::string parameter )  [private]
```

Obtains the Element Parameter parameter from the twiss file.

Parameters

element	name of element in twiss file
parameter	name of element parameter in twiss file

Returns

double that is the value of parameter for element (9999. if parameter or element is not found)

See also

ElementParameterPosition()

4.6.3.2 ElementParameterPosition()

```
int AFPOpticsStudy::OpticsMode::ElementParameterPosition (
            std::string line,
            std::string parameter )  [private]
```

Obtains column numbers for specific parameter.

Decodes the twiss file line defining the names of the parameters in each column This is the line whose first character is ∗

Parameters

line	string containing the line of the file whose first character is ∗
parameter	name of element parameter in twiss file

Returns

column number that corresponds to paramter (-1 if parameter is not found)

4.6.3.3 Initialize()

```
void AFPOpticsStudy::OpticsMode::Initialize (
            std::string twissdir,
            bool isb2,
            double crossing_angle,
            double normex,
            double normey,
            double stprob,
            double dtprob )  [private]
```

SetUp class using inputs.

Runs methods in correct order so that class members are set

Parameters

twissdir	twiss directory path
isb2	boolean to define which beam (true <=> beam2 , false <=> beam1)
crossing_angle	defines crossing angle (in murad)
normex	normalized emittance in x, only used if positive (in m∗rad)
normey	normalized emittance in y, only used if positive (in m∗rad)
stprob	single tag probability
dtprob	double tag probability

See also

ReadTwissFile()

4.6.3.4 List()

```
void AFPOpticsStudy::OpticsMode::List ( ) const
```

Prints values of all member variables to std::cout (except vectors storing the aperture)

See also

ListVal()

4.6.3.5 OpticsParameter_double()

```
double AFPOpticsStudy::OpticsMode::OpticsParameter_double (
            std::string parameter )  [private]
```

Obtains the OpticsParameter parameter from the twiss file.

Parameters

parameter	name of optics parameter in twiss file

Returns

double that is the value of parameter (9999. if parameter is not found)

See also

OpticsParameter_string()

4.6.3.6 OpticsParameter_string()

```
std::string AFPOpticsStudy::OpticsMode::OpticsParameter_string (
            std::string parameter )  [private]
```

Obtains the OpticsParameter parameter from the twiss file.

Optics parameter is a property of full optics, like the beam energy. It is read from the header of the twiss file.

Parameters

parameter	name of optics parameter in twiss file

Returns

string that is the value of parameter ("9999." if parameter is not found)

4.6.3.7 ReadApertureWidths()

```
void AFPOpticsStudy::OpticsMode::ReadApertureWidths ( )  [private]
```

Reads aperture widths and corresponding s-positions.

Assumes m_twiss_file_path is correctly set and collimator widths are already set

4.6.3.8 ReadTwissFile()

```
void AFPOpticsStudy::OpticsMode::ReadTwissFile ( )  [private]
```

Reads twiss file and stores requred values in members.

Assumes m_twiss_dir_path and m_beam are correctly set in order to define m_twiss_file_path

See also

ElementParameter(), OpticsParameter_string(), OpticsParameter_double()

4.6.4 Member Data Documentation

4.6.4.1 m_norm_emittance_x

```
double AFPOpticsStudy::OpticsMode::m_norm_emittance_x  [private]
```

Normalized Emittance [m∗rad].

normalized emittance (both x and y) can be defined directly be the user, but if a negative value is input, then the corresponding normalized emittance is calculated from the twiss file parameters according to:

norm_emittance_{x,y} = beta_rel ∗ gamma_rel ∗ geometric_emittance_{x,y}

The documentation for this class was generated from the following files:

- AFPOpticsStudy/AFPOpticsStudy/OpticsMode.h
- AFPOpticsStudy/src/OpticsMode.cpp

4.7 AFPOpticsStudy::OpticsStudy Class Reference

User interface class to run simulations and obtain plots.

```
#include "AFPOpticsStudy/OpticsStudy.h"
```

Public Member Functions

- OpticsStudy (std::string b1_seq, std::vector< std::string > b1_files, std::string b2_seq, std::vector< std::string > b2_files, double beam_energy, double crossing_angle, double normalized_emittance, std::string prod_dir, AFPStation ∗a_far, AFPStation ∗a_near, AFPStation ∗c_near, AFPStation ∗c_far, std::string description)

  MAD-X Constructor.
- OpticsStudy (std::string twissdir, std::string parentdir, double beam_energy, double crossing_angle, double normalized_emittance, AFPStation ∗a_far, AFPStation ∗a_near, AFPStation ∗c_near, AFPStation ∗c_far, std::string description)

  Twiss Constructor.
- ∼OpticsStudy ()

  Destructor.
- void SetCurrentMode (bool isb2)

  Sets current mode member variable used in methods.
- void SetCurrentDetector (DetIndex idet)

  Sets current detector and all associated current pointers.
- void **SetSingleTagProb** (double st)
- void **SetDoubleTagProb** (double dt)
- void **SetTCL4sigB1** (int n)
- void **SetTCL5sigB1** (int n)
- void **SetTCL4sigB2** (int n)
- void **SetTCL5sigB2** (int n)
- void **SetTCL4mB1** (double w)
- void **SetTCL5mB1** (double w)
- void **SetTCL4mB2** (double w)
- void **SetTCL5mB2** (double w)
- void **SetGeoAccMinE** (double e)
- void **SetGeoAccMinPT** (double p)

- void **SetGeoAccMaxPT** (double p)
- void **SetGeoAccNBinsE** (int n)
- void **SetGeoAccNBinsPT** (int n)
- void **SetGeoAccNEventsPerBin** (int n)
- void **SetColAnaMaxfEl** (double xi)
- void **SetColAnaNEventsPerBin** (int n)
- void **SetColAnaNColSettings** (int n)
- void **SetColAnaNFracELossBins** (int n)
- void **SetColAnaRebin** (int n)
- void **SetTagTimMinPileup** (double mu)
- void **SetTagTimMaxPileup** (double mu)
- void **SetTagTimToFSettings** (std::vector< double > v)
- void **SetTagTimEventsMultiplier** (double n)
- void **SetProPosLowE** (double e)
- void **SetProPosLowPT** (double p)
- void **SetProPosHighPT** (double p)
- void **SetProPosNAngles** (int n)
- void **SetTrajExObsPosStep** (double s)
- void **SetTrajExObsPosMax** (double s)
- void **SetTrajExFracELossOpts** (std::vector< double > v)
- void **SetTrajExTransversePOpts** (std::vector< double > v)
- void **SetParametFracEnergyLossMax** (double xi)
- void **SetParametXRange** (double center, double range)
- void **SetParametYRange** (double center, double range)
- void **SetParametZRange** (double center, double range)
- void **SetParametSXRange** (double center, double range)
- void **SetParametSYRange** (double center, double range)
- void **SetParametGridDimension** (int n)
- void **SetParametMinPolDeg** (int d)
- void **SetParametMaxPolDeg** (int d)
- void **SetParametMinPolVal** (double min)
- void **SetParametMinCoefVal** (double min)
- void **SetParametNEvents** (int n)
- void **SetParametCutEvent** (int n)
- void **SetThinFloorNEvents** (int n)
- void PrintCurrentPointers () const

  Prints information about the current afp and mode to std::cout.
- void GeometricAcceptance () const

  Produces a geometric acceptance plot for the current afp and saves it into the geometric acceptance directory.
- void CollimatorAnalysis () const

  Produces plots displaying how the acceptace at the current afp changes depending on the settings for the TCL4 and TCL5 collimators.
- void ProtonPositions () const

  Produces a plot of example positions of simulated protons at the current afp detector plane.
- void TrajectoryExamples (bool draw_aperture) const

  Produces plots of example trajectories of protons with different intial energy and momenta for the current afp.
- void TagTimeProbabilities () const

  Produces plots (lin-log and log-log) of the probabilities of protons being tagged and timed depending on the pileup.
- void FloorLengthProbabilities () const

  Produces plots of the probability of proton going through the current afp floor for a given length and how the proton passes through the floor.
- void FindParameterisation (bool validate)

  Runs parameterisation function for member variable corresponding to current afp.
- void ValidateDegrees ()

Determines the degree validity of the current parameterisation.

- void PrintSpecificParameterisation (const std::vector< std::vector< int > > &degrees) const

  Prints specific parameterisation to file given degrees input by user.
- void PrintPreciseParameterisation () const

  Prints most precise parameterisation according the degree validity determined by the program.
- void ValidateSpecificParameterisation (const std::vector< std::vector< int > > &degrees) const

  Validates specific parameterisation by testing it against simulations (obtains SpecificParam from current parameterisation and degrees input by user)
- void ValidateSpecificParameterisation (const std::string &parameterisation_file) const

  Validates specific parameterisation by testing it against simulations (obtains SpecificParam from reading the parameterisation file)
- void ValidatePreciseParameterisation () const

  Validates precise parameterisation by testing it against simulations.

Private Member Functions

- void Simulators ()

  Initializes the four simulators according to the member variables used by Simulator (must be re-run if any details of the member variables change - i.e. distance to AFP from IP)
- std::string CreateDirectory (std::string dir_name)

  Creates a directory with name dir_name inside of parent_dir and inside of root storage file.
- void PrintingDirectories ()

  Creates all of the directories for printing the plots and saving the histograms.
- std::string BeamInfoString () const

  Creates a string that has the current beam energy, crossing angle, and description.
- std::string DetectorPlotPrefix () const

  Creates a prefix string for plots describing specific afp stations.
- std::string ArmPlotPrefix () const

  Creates a prefix string for plots describing a specific arm.
- void Initialize (std::string twissdir, std::string parentdir, double beam_energy, double crossing_angle, double normalized_emittance, AFPStation *a_far, AFPStation *a_near, AFPStation *c_near, AFPStation *c_far, std::string description)

  Initializes the class.
- void SetPrimSec (int prim, int sec)

  Set primary and secondary current pointers.
- std::string ProgressBar (const double percentage) const

  Constructs a string that is the "progress bar" for the input percentage.
- void PrintParameterisationHeader (std::ostream &out) const

  Prints the header for the parameterisation files to out.
- void Validate (const SpecificParam &sp) const

  Generates histograms comparing the outputs of parameterisation and simulation.

Private Attributes

- const bool s_BEAM_1 = false

  beam label
- const bool s_BEAM_2 = true

  beam label
- const int s_iA_FAR = 0

  afp station index

- const int s_iA_NEAR = 1
  afp station index
- const int s_iC_NEAR = 2
  afp station index
- const int s_iC_FAR = 3
  afp station index
- OpticsMode m_mode_b1
  mode for beam1
- OpticsMode m_mode_b2
  mode for beam2
- AFPStation * m_afp [4]
  array of pointers to afp stations
- GeoAccPars m_gap
  geometric acceptance simulation parameters
- ColAnaPars m_cap
  collimator analysis simulation parameters
- TagTimPars m_ttp
  tagging and timing probabilities simulation parameters
- ProPosPars m_prop
  proton position simulation parameters
- TrajExPars m_tep
  example trajectories simulation parameters
- ParametPars m_parp
  parameterisation parameters
- int m_ThinFloor_num_events
  number of events to simulate for thin floor
- Simulator m_sim [4]
  array of Simulators (one for each afp station)
- Parameterisation m_param [4]
  array of Parameterisations (one for each afp station)
- std::string m_description
  description to be added to the parent directory and each plot
- std::string m_parent_dir
  filepath to directory containing product folders
- std::string m_twiss_dir
  filepath to directory containing twiss files
- TFile * m_storagefile
  pointer to root file that stores histograms
- std::string m_geometricacceptance_dir
  filepath to geometric acceptance directory
- std::string m_collimatoranalysis_dir
  filepath to collimator analysis directory
- std::string m_pileupprediction_dir
  filepath to tagging and timing probabilities directory
- std::string m_protonpositions_dir
  filepath to proton positions directory
- std::string m_trajectoryexamples_dir
  filepath to example trajectories directory
- std::string m_thinfloorhits_dir
  filepath to thin floor hits directory
- std::string m_parameterisation_dir

filepath to parameterisation directory

- OpticsMode * m_curr_mode

  pointer to current mode
- AFPStation * m_curr_prim_afp

  pointer to current primary afp
- AFPStation * m_curr_sec_afp

  pointer to current secodnary afp (other station on the same side as primary)
- Simulator * m_curr_sim

  pointer to current Simulator (matching primary afp)
- Parameterisation * m_curr_param

  pointer to current Parameterisation (matching primary afp)

4.7.1 Detailed Description

User interface class to run simulations and obtain plots.

This class uses an OpticsMode instance for each beam, and AFPStation, Simulator, and Parameterisation instances for each detector. These member classes are mainly used for storage of the different settings (OpticsMode and AFPStation) or running functions that would be too complicated or extensive to be put into this class (Simulator and Parameterisation). All of the other members are simply used to make the methods easier to operate. The simulation parameter classes (GeoAccPars, ColAnaPars, TagTimPars, ProPosPars, TrajExPars, ParametPars) are mainly there to keep the details of how to run the simulations in a nice package. The pointers beginning with "curr" are used to point to the "current" member that a method should use. Finally, the other members are used to make storage of the different plots and histograms easy and efficient. The only classes used in this framework that aren't members of this class are the canvas classes (GeoAccCanvas, ColAnaCanvas, PilPreCanvas, ProPosCanvas, and TrajExCanvas). These are not members because they are only invoked to draw and print the various plots at the end of some methods. (However, it would be rather simple to incorporate these classes as members if it becomes more convenient in the future).

4.7.2 Constructor & Destructor Documentation

4.7.2.1 OpticsStudy() [1/2]

```
AFPOpticsStudy::OpticsStudy::OpticsStudy (
            std::string b1_seq,
            std::vector< std::string > b1_files,
            std::string b2_seq,
            std::vector< std::string > b2_files,
            double beam_energy,
            double crossing_angle,
            double normalized_emittance,
            std::string prod_dir,
            AFPStation * a_far,
            AFPStation * a_near,
            AFPStation * c_near,
            AFPStation * c_far,
            std::string description )
```

MAD-X Constructor.

Uses inputs and WriteTwiss class to construct twiss files. Then uses newly constructed twiss files to initialize the member variables of the class.

Parameters

b1_seq	filepath to madx sequence file for beam1
b1_files	vector of filepaths to other madx files required for beam1
b2_seq	filepath to madx sequence file for beam2
b2_files	vector of filepaths to other madx files required for beam2
beam_energy	energy of beam in GeV
crossing_angle	crossing angle at IP in murad
normalized_emittance	normalized emittance at IP in m∗rad
prod_dir	filepath to directory in which products should be saved (blank means use current directory)
a_far	pointer to AFPStation for A FAR
a_near	pointer to AFPStation for A NEAR
c_near	pointer to AFPStation for C NEAR
c_far	pointer to AFPStation for C FAR

See also

Initialize(), WriteTwiss::WriteTwissFile()

4.7.2.2 OpticsStudy() [2/2]

```
AFPOpticsStudy::OpticsStudy::OpticsStudy (
            std::string twissdir,
            std::string parentdir,
            double beam_energy,
            double crossing_angle,
            double normalized_emittance,
            AFPStation * a_far,
            AFPStation * a_near,
            AFPStation * c_near,
            AFPStation * c_far,
            std::string description )
```

Twiss Constructor.

Assumes twiss files are already written correctly and uses input to initialize class

Parameters

twissdir	filepath to directory of twiss files
parentdir	filepath to directory in which plots should be printed (blank means use current directory)
beam_energy	energy of beam in GeV
crossing_angle	crossing angle at IP in murad
normalized_emittance	normalized emittance at IP in m∗rad
a_far	pointer to AFPStation for A FAR
a_near	pointer to AFPStation for A NEAR
c_near	pointer to AFPStation for C NEAR
c_far	pointer to AFPStation for C FAR

See also

Initialize()

4.7.2.3 ~OpticsStudy()

```
AFPOpticsStudy::OpticsStudy::~OpticsStudy ( )
```

Destructor.

Writes storage file before deleting class.

4.7.3 Member Function Documentation

4.7.3.1 ArmPlotPrefix()

```
std::string AFPOpticsStudy::OpticsStudy::ArmPlotPrefix ( ) const  [private]
```

Creates a prefix string for plots describing a specific arm.

Takes all information from the current mode. Ends in "_".

See also

BeamInfoString()

4.7.3.2 CollimatorAnalysis()

```
void AFPOpticsStudy::OpticsStudy::CollimatorAnalysis ( ) const
```

Produces plots displaying how the acceptace at the current afp changes depending on the settings for the TCL4 and TCL5 collimators.

See also

DetectorPlotPrefix(), Simulator::Collimator(), ProgressBar(), ColAnaCanvas::ColAnaCanvas(), ColAnaCanvas::DrawClosedOpen(), ColAnaCanvas::DrawMaxfEl(), ColAnaCanvas::Save()

4.7.3.3 CreateDirectory()

```
std::string AFPOpticsStudy::OpticsStudy::CreateDirectory (
            std::string dir_name )  [private]
```

Creates a directory with name dir_name inside of parent_dir and inside of root storage file.

Parameters

dir_name	name of the directory to be created (no need for any slashes before or after)

Returns

full filepath (as a string) to created directory (returns parent directory if unable to create new directory)

4.7.3.4 DetectorPlotPrefix()

```
std::string AFPOpticsStudy::OpticsStudy::DetectorPlotPrefix ( ) const  [private]
```

Creates a prefix string for plots describing specific afp stations.

Takes all information from the current mode and afp station pointers. Ends in "_" so that the name of the plot can be directly concatenated

See also

BeamInfoString()

4.7.3.5 FindParameterisation()

```
void AFPOpticsStudy::OpticsStudy::FindParameterisation (
            bool validate )
```

Runs parameterisation function for member variable corresponding to current afp.

If run on an already determined parameterisation, will recalculate the coefficients, but not the polynomial values. Prints polynomial value and coefficient tables to file. FindParameterisation(true); is equivalent to FindParameterisation(false); ValidateDegrees();

Parameters

validate	input to tell program whether or not to determine validity of different degree options

See also

ValidateDegrees(), Parameterisation::Run()

4.7.3.6 FloorLengthProbabilities()

```
void AFPOpticsStudy::OpticsStudy::FloorLengthProbabilities ( ) const
```

Produces plots of the probability of proton going through the current afp floor for a given length and how the proton passes through the floor.

A class to perform the styling and printing of these plots has not been written yet, so all of the styling is done directly in this function.

One of the histograms produced by this function gives the probabilities of different "situations". These situations are explained by the image included in the repository.

See also

Simulator::FloorPathLength()

4.7.3.7 GeometricAcceptance()

```
void AFPOpticsStudy::OpticsStudy::GeometricAcceptance ( ) const
```

Produces a geometric acceptance plot for the current afp and saves it into the geometric acceptance directory.

See also

DetectorPlotPrefix(), Simulator::Acceptance(), ProgressBar(), GeoAccCanvas::GeoAccCanvas(), GeoAccCanvas::Draw(), GeoAccCanvas::Save()

4.7.3.8 Initialize()

```
void AFPOpticsStudy::OpticsStudy::Initialize (
            std::string twissdir,
            std::string parentdir,
            double beam_energy,
            double crossing_angle,
            double normalized_emittance,
            AFPStation * a_far,
            AFPStation * a_near,
            AFPStation * c_near,
            AFPStation * c_far,
            std::string description )  [private]
```

Initializes the class.

Sets member variables to corresponding inputs and constructs all other member variables using this information.

See also

Simulators(), PrintingDirectories(), SetCurrentDetector()

4.7.3.9 PrintingDirectories()

```
void AFPOpticsStudy::OpticsStudy::PrintingDirectories ( )  [private]
```

Creates all of the directories for printing the plots and saving the histograms.

See also

CreateDirectory()

4.7.3.10 PrintParameterisationHeader()

```
void AFPOpticsStudy::OpticsStudy::PrintParameterisationHeader (
            std::ostream & out ) const  [private]
```

Prints the header for the parameterisation files to out.

The parameterisation files have a traditional header that does not match the specific nameing convections of this framework. This function prints a header matching the previous definitions, so that parameterisation files can be compatible with other programs.

4.7.3.11 PrintPreciseParameterisation()

```
void AFPOpticsStudy::OpticsStudy::PrintPreciseParameterisation ( ) const
```

Prints most precise parameterisation according the degree validity determined by the program.

Pulls coefficients from Parameterisation member, so this must be run after FindParameterisation(). Will only print most precise parameterisation if degrees have been validated.

See also

DetectorPlotPrefix(), PrintParameterisationHeader(), SpecificParam::Print(), Parameterisation::SpecificParameterisation()

4.7.3.12 PrintSpecificParameterisation()

```
void AFPOpticsStudy::OpticsStudy::PrintSpecificParameterisation (
            const std::vector< std::vector< int > > & degrees ) const
```

Prints specific parameterisation to file given degrees input by user.

Pulls coefficients from Parameterisation member, so this must be run after FindParameterisation(). Input vector has indices: (which observable , which polynomial)

Parameters

degrees	two-dimensional vector defining the degrees of each polynomial

See also

DetectorPlotPrefix(), PrintParameterisationHeader(), SpecificParam::Print()

4.7.3.13 ProgressBar()

```
std::string AFPOpticsStudy::OpticsStudy::ProgressBar (
            const double percentage ) const  [private]
```

Constructs a string that is the "progress bar" for the input percentage.

The string is one line that has a carriage return () at the end so that when this string is output to std::cout with std::flush after, it replaces itself at each print.

Parameters

percentage	current progress of running process in %

Returns

string that represents a loading bar

4.7.3.14 ProtonPositions()

```
void AFPOpticsStudy::OpticsStudy::ProtonPositions ( ) const
```

Produces a plot of example positions of simulated protons at the current afp detector plane.

See also

Simulator::DetectorPosition(), ProPosCanvas::ProPosCanvas(), ProPosCanvas::Draw(), ProPosCanvas::Save()

4.7.3.15 SetCurrentDetector()

```
void AFPOpticsStudy::OpticsStudy::SetCurrentDetector (
            DetIndex idet )
```

Sets current detector and all associated current pointers.

Uses a switch statement between the different possibilities of the enum class DetIndex.

See also

SetPrimSec()

4.7.3.16 SetCurrentMode()

```
void AFPOpticsStudy::OpticsStudy::SetCurrentMode (
            bool isb2 )
```

Sets current mode member variable used in methods.

(true <=> beam2 , false <=> beam1)

See also

SetPrimSec()

4.7.3.17 Simulators()

```
void AFPOpticsStudy::OpticsStudy::Simulators ( )  [private]
```

Initializes the four simulators according to the member variables used by Simulator (must be re-run if any details of the member variables change - i.e. distance to AFP from IP)

Uses the Simulator constructor to make the beamlines that the simulations are run on. (See the Simulator class for details).

4.7.3.18 TagTimeProbabilities()

```
void AFPOpticsStudy::OpticsStudy::TagTimeProbabilities ( ) const
```

Produces plots (lin-log and log-log) of the probabilities of protons being tagged and timed depending on the pileup.

See also

Simulator::TagTimeProbs(), PilPreCanvas::PilPreCanvas(), PilPreCanvas::Draw(), PilPreCanvas::Save()

4.7.3.19 TrajectoryExamples()

```
void AFPOpticsStudy::OpticsStudy::TrajectoryExamples (
            bool draw_aperture ) const
```

Produces plots of example trajectories of protons with different intial energy and momenta for the current afp.

See also

Simulator::TrajectoryPosition(), TrajExCanvas::TrajExCanvas(), TrajExCanvas::Draw(), TrajExCanvas::Save()

4.7.3.20 Validate()

```
void AFPOpticsStudy::OpticsStudy::Validate (
            const SpecificParam & sp ) const  [private]
```

Generates histograms comparing the outputs of parameterisation and simulation.

Writes histograms and prints them without styling. These histograms are simple distributions of the absolute value of the difference between detector kinematic values and the values predicted by simulation.

Parameters

sp	a SpecificParam that is already set to have specific coefficients

See also

Simulator::DetectorKinematics(), SpecificParam::Eval(), ProgressBar()

4.7.3.21 ValidateDegrees()

```
void AFPOpticsStudy::OpticsStudy::ValidateDegrees ( )
```

Determines the degree validity of the current parameterisation.

If run on an already validated parameterisation, will recalculate validities after printing a warning. Prints degree validity table to file. The details of the method for determining the validity of degree options is detailed in the Parameterisation class.

See also

Parameterisation::SetDegreeValidity(), Parameterisation::PrintDegreeValidity()

4.7.3.22 ValidatePreciseParameterisation()

```
void AFPOpticsStudy::OpticsStudy::ValidatePreciseParameterisation ( ) const
```

Validates precise parameterisation by testing it against simulations.

Pulls coefficients from Parameterisation member, so this must be run after FindParameterisation().

See also

Parameterisation::SpecificParameterisation(), Validate()

4.7.3.23 ValidateSpecificParameterisation() [1/2]

```
void AFPOpticsStudy::OpticsStudy::ValidateSpecificParameterisation (
            const std::vector< std::vector< int > > & degrees ) const
```

Validates specific parameterisation by testing it against simulations (obtains SpecificParam from current parameterisation and degrees input by user)

Pulls coefficients from Parameterisation member, so this must be run after FindParameterisation(). Input vector has indices: (which observable , which polynomial)

Parameters

degrees	two-dimensional vector used to tell the program which degrees to use for each polynomial

See also

DetectorPlotPrefix(), Parameterisation::SpecificParametersiation(), Validate()

4.7.3.24 ValidateSpecificParameterisation() [2/2]

```
void AFPOpticsStudy::OpticsStudy::ValidateSpecificParameterisation (
            const std::string & parameterisation_file ) const
```

Validates specific parameterisation by testing it against simulations (obtains SpecificParam from reading the parameterisation file)

Pulls coefficients from input file, so this can be run at anytime.

See also

SpecificParam::Load, Validate()

The documentation for this class was generated from the following files:

- AFPOpticsStudy/AFPOpticsStudy/OpticsStudy.h
- AFPOpticsStudy/src/OpticsStudy.cpp

4.8 AFPOpticsStudy::Parameterisation Class Reference

Calculates polynomial values and parameterisation coefficients.

```
#include <Parameterisation.h>
```

Public Member Functions

- Parameterisation ()
  Default constructor.
- Parameterisation (OpticsMode *mode, AFPStation *afp, Simulator *sim, ParametPars *parp)
  SetUp constructor.
- Parameterisation (OpticsMode *mode, AFPStation *afp, Simulator *sim, ParametPars *parp, bool run)
  Run constructor.
- ~Parameterisation ()
  Destructor.
- void SetUp (OpticsMode *mode, AFPStation *afp, Simulator *sim, ParametPars *parp)
  Sets the member pointers to corresponding input objects.
- void Run ()

Runs parameterisation after checking that everything is correctly set up.

- void ReCalculate ()

  Recalculates both the polynomial values and coefficients no matter what was previously there (erases previously stored information before preceding)
- void ValidateDegrees ()

  Validates all of the possible combinations of degrees against the simulation.
- void PrintFullPolyValsTable (std::ostream &out) const

  print out full polynomial values table
- void PrintFullCoeffsTable (std::ostream &out) const

  print out full coefficients table
- void PrintDegreeValidity (std::ostream &out) const

  Prints out table of observables containing different degree options and their corresponding validity.
- SpecificParam SpecificParameterisation (std::string par_name, const std::vector< std::vector< int > > &degrees) const

  sets specific parameterisation by copying coefficient vectors for input degrees
- SpecificParam SpecificParameterisation () const

  Sets specific parameterisation by obtaining the degrees specified by m_precise_degrees if they have been determined, if not, returns a default parameterisation (all degrees are set to the maximum, except G and H degreees are set to zero)

Private Member Functions

- void InitializeTables ()

  Constructs storage tables in the right dimensions according to members stored in ParametPars.
- bool SetPolynomialVals ()

  Obtains the values for the polynomials A,...,H for each of the given observables.
- bool SetCoefficients ()

  Calculates values of the coefficients for polynomials A,...,H for each of the given observables and the range of degrees in m_parp.
- double EvalPolynomial (int iobs, int ipol, int ideg, double fEl) const

  calculates value of polynomial ipol for value fEl and degree ideg
- bool SetDegreeValidity ()

  Calculates validity of the different degrees of polynomials.
- void SetPreciseDegrees ()

  Determines the most precise set of degrees from degree validity.

Private Attributes

- OpticsMode * m_mode

  current mode
- AFPStation * m_afp

  current afp station
- Simulator * m_sim

  current simulator
- ParametPars * m_parp

  parameterisation parameters
- std::vector< std::vector< std::vector< double > > > m_polynomial_vals
- bool m_poly_vals_filled

  true if m_polynomial_vals has been successfully determined
- std::vector< std::vector< double > > m_fracEloss_vals

- std::vector< std::vector< std::vector< std::vector< double > > > > m_coefficients
- bool m_coeffs_filled

  true if m_coefficients has been successfully determined
- std::vector< std::vector< std::vector< int > > > m_degree_options
- std::vector< std::vector< double > > m_degree_validity

  Validity values for the degree options. Indices are (which observalbe , validity measure of corresponding degree option)
- bool m_deg_valid_filled

  true if degree validity has been determined for each of the degree options
- std::vector< std::vector< int > > m_precise_degrees

  vector containing the most precise degrees as determined by degree validity

Static Private Attributes

- static const int s_iA = 0

  polynomial index
- static const int s_iB = 1

  polynomial index
- static const int s_iC = 2

  polynomial index
- static const int s_iD = 3

  polynomial index
- static const int s_iE = 4

  polynomial index
- static const int s_iF = 5

  polynomial index
- static const int s_iG = 6

  polynomial index
- static const int s_iH = 7

  polynomial index
- static const int s_iX = 0

  detector observable index
- static const int s_iY = 1

  detector observable index
- static const int s_iSX = 2

  detector observable index
- static const int s_iSY = 3

  detector observable index

4.8.1 Detailed Description

Calculates polynomial values and parameterisation coefficients.

This parameterisation method assumes that the values of the proton kinematics at the detector can be covered by small deviations away from the zeros of the IP kinematic values. More specifically, if x is any obersvable at the detector and xi is the fractional energy loss, then this class determines the coefficients of the polynomials in

x = A(xi)+xIP∗B(xi)+yIP∗C(xi)+zIP∗D(xi)+sxIP∗E(xi)+syIP∗F(xi)+zIP∗sxIP∗G(xi)+zIP∗syIP∗H(xi)

See Appendix A of the AFP TDR for more information about this transport parameterisation.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Parameterisation() [1/3]

```
AFPOpticsStudy::Parameterisation::Parameterisation ( )
```

Default constructor.

Sets member booleans to false and does nothing else

4.8.2.2 Parameterisation() [2/3]

```
AFPOpticsStudy::Parameterisation::Parameterisation (
            OpticsMode * mode,
            AFPStation * afp,
            Simulator * sim,
            ParametPars * parp )
```

SetUp constructor.

Takes in arguments, sets up tables, and calls Default constructor

See also

SetUp(), InitializeTables()

4.8.2.3 Parameterisation() [3/3]

```
AFPOpticsStudy::Parameterisation::Parameterisation (
            OpticsMode * mode,
            AFPStation * afp,
            Simulator * sim,
            ParametPars * parp,
            bool run )
```

Run constructor.

Calls SetUp constructor and runs parameterisation if run==true Identical to SetUp constructor if run==false

See also

SetUp(), InitializeTables(), Run()

4.8.3 Member Function Documentation

4.8.3.1 ReCalculate()

```
void AFPOpticsStudy::Parameterisation::ReCalculate ( )
```

Recalculates both the polynomial values and coefficients no matter what was previously there (erases previously stored information before preceding)

See also

InitializeTables(), SetCoefficients()

4.8.3.2 Run()

```
void AFPOpticsStudy::Parameterisation::Run ( )
```

Runs parameterisation after checking that everything is correctly set up.

If Run() is called on an already filled parameterisation, it will recalculate the coefficients from the polynomial values, but will not recalculate the polynomial values.

See also

SetCoefficients()

4.8.3.3 SetCoefficients()

```
bool AFPOpticsStudy::Parameterisation::SetCoefficients ( )  [private]
```

Calculates values of the coefficients for polynomials A,...,H for each of the given observables and the range of degrees in m_parp.

Performed after determining the values of the various polynomials for each value of the fractional energy loss. This function does a simple polynomial fit of each of these polynomials using each of the different degrees stored in ParametPars.

If max is the maximum polynomial degree and min is the minimum polynomial degree, this function has $4*8*(max - min + 1)$ polynomial fits in it.

Returns

true if successfully obtains values for all coefficients.

See also

SetPolynomialVals()

4.8.3.4 SetPolynomialVals()

```
bool AFPOpticsStudy::Parameterisation::SetPolynomialVals ( )  [private]
```

Obtains the values for the polynomials A,...,H for each of the given observables.

Gets these values from keeping all inputs at their center except the ones corresponding to the different polynomials (e.g. B only has xIP not at center). For each polynomial, the fractional energy loss is varied and for each value of fractional energy loss, the coefficient variable is iterated through its range. The value of the polynomial at a specific fractional energy loss is then taken to be the slope of the linear fit between the coefficient variable and the detector observable. For example, to obtain the value of the polynomial B(xi) at a specific xi, xIP is iterated through its entire range, and then for each detector observable obs, one finds the linear fit obs = a∗xIP + b and sets B(xi) to the value a.

This function carries a factor of 1/(1-xi) in places where the center of the sxIP and syIP ranges are used. This factor is there in order for this parameterisation method to match the previous parameterisation method. If this factor is removed, make sure to remove it also in the degree validation function (SetDegreeValidity()) and in the parameterisation validation function (OpticsStudy::Validate(SpecificParam)).

If d is the grid dimension, then there are d+5d^2+2d^3 simulations and 28 linear fits in this function.

Returns

true if successfully obtains values for all polynomials.

A //

B,...,F //

G,H //

4.8.3.5 SetPreciseDegrees()

```
void AFPOpticsStudy::Parameterisation::SetPreciseDegrees ( )  [private]
```

Determines the most precise set of degrees from degree validity.

Simply takes the degree option for each observable that returns the highest value of degree validity. Clears whatever used to be stored.

4.8.3.6 ValidateDegrees()

```
void AFPOpticsStudy::Parameterisation::ValidateDegrees ( )
```

Validates all of the possible combinations of degrees against the simulation.

See also

SetDegreeValidity()

4.8.4 Member Data Documentation

4.8.4.1 m_coefficients

```
std::vector< std::vector< std::vector< std::vector<double> > > > AFPOpticsStudy::Parameterisation↩
::m_coefficients [private]
```

Values of coefficients of polynomials for given observables. Indices are: (which observalbe , which polynomial , degree of polynomial , index of coefficient)

4.8.4.2 m_degree_options

```
std::vector< std::vector< std::vector<int> > > AFPOpticsStudy::Parameterisation::m_degree_↩
options [private]
```

Validity of the different degrees (a higher percentage means more values of the parameterisation with those degrees were valid). Indices are: (which observable , which set of degrees , which polynomial)

4.8.4.3 m_fracEloss_vals

```
std::vector< std::vector<double> > AFPOpticsStudy::Parameterisation::m_fracEloss_vals [private]
```

Values of fractional energy loss that polynomials A,...,H use Will be identical to FracEnergyLossRange in ParametPars if no protons are lost. Indices are: (which polynomial , index of value of fractional energy loss)

4.8.4.4 m_polynomial_vals

```
std::vector< std::vector< std::vector<double> > > AFPOpticsStudy::Parameterisation::m_↩
polynomial_vals [private]
```

Values of the polynomials A,...,H for a given observable. Indices are: (which observable , which polynomial , index of value of fractional energy loss)

The documentation for this class was generated from the following files:

- AFPOpticsStudy/AFPOpticsStudy/parameterisation/Parameterisation.h
- AFPOpticsStudy/src/parameterisation/Parameterisation.cxx

4.9 AFPOpticsStudy::ParametPars Class Reference

Storage of parameters used in determination of transport parameterisation coefficients.

```
#include <ParametPars.h>
```

Public Member Functions

- ParametPars ()

  Default Constructor.
- double **FracEnergyLoss** (int i) const
- double **X** (int i) const
- double **XCenter** () const
- double **Y** (int i) const
- double **YCenter** () const
- double **Z** (int i) const
- double **ZCenter** () const
- double **SX** (int i) const
- double **SXCenter** () const
- double **SY** (int i) const
- double **SYCenter** () const
- int **GridDimension** () const
- int **MinPolDeg** () const
- int **MaxPolDeg** () const
- double **MinPolVal** () const
- double **MinCoefVal** () const
- int **NEvents** () const
- int **CutEvent** () const
- void **SetFracEnergyLossMax** (double max)
- void **SetXRange** (double center, double range)
- void **SetYRange** (double center, double range)
- void **SetZRange** (double center, double range)
- void **SetSXRange** (double center, double range)
- void **SetSYRange** (double center, double range)
- void **SetGridDimension** (int n)
- void **SetMinPolDeg** (int n)
- void **SetMaxPolDeg** (int n)
- void **SetMinPolVal** (double v)
- void **SetMinCoefVal** (double v)
- void **SetNEvents** (int n)
- void **SetCutEvent** (int n)

Private Member Functions

- void SetCenterRange (const int val, double center, double range)

  sets center and range for value val
- double CalculateVal (const int val, int i) const

  calculates value for val at index i
- void SetDefaultRanges ()

  sets ranges to defaults (used in constructors)

Private Attributes

- const int s_X = 0

  index for storage vector
- const int s_Y = 1

  index for storage vector
- const int s_Z = 2

  index for storage vector
- const int s_SX = 3

  index for storage vector
- const int s_SY = 4

  index for storage vector
- std::vector< double > m_center

  List of 'centers' of initial kinematic ranges (usually zero)
- std::vector< double > m_range

  List of ranges of initial kinematics.
- double m_maxfEl

  Maximum fractional energy loss (minimum is zero)
- int m_grid_dimension

  Dimension of the grid in parameter space.
- int m_minpoldeg

  Minimum polynomial degree to determine coefficients for.
- int m_maxpoldeg

  Maximum polynomial degree to determine coefficeints for.
- double m_minpolval

  Minimum value for polynomials B,...,H (any value less than it is set to zero)
- double m_mincoeffval

  Minimum value for coefficients (any value less than it is set to zero)
- int m_nevents

  Number of events to be used in validation.
- int m_cutevent

  number of events to simulate before cutting out degree options with less than 80% validity

4.9.1 Detailed Description

Storage of parameters used in determination of transport parameterisation coefficients.

This class also handles calculating the different inputs for the simulation during the training of the parameterisation. Each input iterates through the range [center-range , center+range], where each input can have a different value for center and different value for range. This range is cut into a grid dimension number of divisions, so the higher the grid dimension, the more finely the input's range is sampled.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 ParametPars()

```
AFPOpticsStudy::ParametPars::ParametPars ( )
```

Default Constructor.

sets member variables to defaults: vector of centers = { 0 , 0 , 0 , 0 , -170e-6 } vector of ranges = { 50e-6 , 50e-6 , 0.05 , 2e-4 , 2e-4 } maximum fractional energy loss = 1500/6500 grid dimension = 60 minimum polynomial degree = 3 maximum polynomial degree = 5 minimum polynomial value = 1e-8 minimum coefficient value = 1e-10

See also

SetDefaultRanges()

The documentation for this class was generated from the following files:

- AFPOpticsStudy/AFPOpticsStudy/parameterisation/ParametPars.h
- AFPOpticsStudy/src/parameterisation/ParametPars.cxx

4.10 AFPOpticsStudy::PilPreCanvas Class Reference

printing and drawing probabilities that depend on pileup

```
#include <PilPreCanvas.h>
```

Public Member Functions

- PilPreCanvas ()
  Default Constructor.
- PilPreCanvas (const OpticsMode *mode, const TagTimPars *ttp)
  Basic Constructor.
- ~PilPreCanvas ()
  Destructor.
- void Draw (std::vector< TH1F *> &hists, int ylog, int xlog)
  draws histograms onto canvas scaling the axes according to the inputs
- void Save (std::string filepath)
  saves canvas to filepath

Private Member Functions

- void SetROOTStyle ()
  set global printing and drawing style
- void BuildLabels ()
  build plot labels
- void DeleteLabels ()
  delete plot labels

Private Attributes

- const OpticsMode * m_mode
  current mode
- const TagTimPars * m_ttp
  Tagging and Timing simulation parameters.
- TCanvas * m_pilprecanvas
  canvas drawings will be printed to
- TGraph * m_frame
  frame to force certain graph size
- TLatex * m_beaminfo
  beam information taken from current mode
- TLegend * m_legend
  legend for the different histograms

4.10.1 Detailed Description

printing and drawing probabilities that depend on pileup

4.10.2 Constructor & Destructor Documentation

4.10.2.1 PilPreCanvas() [1/2]

```
AFPOpticsStudy::PilPreCanvas::PilPreCanvas ( )
```

Default Constructor.

sets global style and initializes canvas

See also

SetROOTStyle()

4.10.2.2 PilPreCanvas() [2/2]

```
AFPOpticsStudy::PilPreCanvas::PilPreCanvas (
            const OpticsMode * mode,
            const TagTimPars * ttp )
```

Basic Constructor.

Sets member variables to corresponding inputs, sets global style, initializes canvas, and builds plot labels

See also

SetROOTStyle(), BuildLabels()

4.10.2.3 ~PilPreCanvas()

```
AFPOpticsStudy::PilPreCanvas::~PilPreCanvas ( )
```

Destructor.

See also

DeleteLabels()

4.10.3 Member Function Documentation

4.10.3.1 Draw()

```
void AFPOpticsStudy::PilPreCanvas::Draw (
            std::vector< TH1F *> & hists,
            int ylog,
            int xlog )
```

draws histograms onto canvas scaling the axes according to the inputs

It is assumed that the title of the histogram should be the label of it in the legend.

Parameters

hists	vector of TH1F histograms that stores the probability values
ylog	defining scale of y axis (0 <=> linear, 1 <=> logarithmic)
xlog	defining scale of x axis

The documentation for this class was generated from the following files:

- AFPOpticsStudy/AFPOpticsStudy/canvases/PilPreCanvas.h
- AFPOpticsStudy/src/canvases/PilPreCanvas.cpp

4.11 AFPOpticsStudy::ProPosCanvas Class Reference

drawing and printing of example proton positions

```
#include <ProPosCanvas.h>
```

Public Member Functions

- ProPosCanvas ()
  Default Constructor.

- ProPosCanvas (const OpticsMode *mode, const AFPStation *afp, const ProPosPars *prop)

  Basic Constructor.
- ~ProPosCanvas ()

  Destructor.
- void Draw (std::vector< FPTracker::Point > points[ ][2])

  draw final proton positions onto canvas with labels
- void Save (std::string filepath)

  saves canvas to filepath

Private Member Functions

- void SetROOTStyle ()

  set global drawing and printing style
- void BuildLabels ()

  build labels that depend no current mode and afp station
- void DeleteLabels ()

  delete labels

Private Attributes

- const OpticsMode * m_mode

  current mode
- const AFPStation * m_afp

  current afp station
- const ProPosPars * m_prop

  example proton position parameters
- TCanvas * m_proposcanvas

  drawing and printing canvas
- TGraph * m_frame_graph

  graph to draw first so that frame is the correct size
- TGraph * m_graph_point [2][2]

  graphs of the example proton positions
- TEllipse * m_beam_pipe

  graph of beam pipe
- TLegend * m_legend

  legend for different graphs
- TBox * m_AFPBox

  plot of afp station's active region
- TLatex * m_energy

  label about beam energy
- TLatex * m_beta

  label about beta function
- TLatex * m_cross_angle

  label about crossing angle
- TLatex * m_xdist

  label about afp station's x separation from beam center
- TLatex * m_obs_place

  label about afp station's distance from IP
- TLine * m_vertline

  vertical line separating legend and other information

4.11.1 Detailed Description

drawing and printing of example proton positions

4.11.2 Constructor & Destructor Documentation

4.11.2.1 ProPosCanvas() [1/2]

```
AFPOpticsStudy::ProPosCanvas::ProPosCanvas ( )
```

Default Constructor.

sets global style and initializes canvas

See also

SetROOTStyle()

4.11.2.2 ProPosCanvas() [2/2]

```
AFPOpticsStudy::ProPosCanvas::ProPosCanvas (
            const OpticsMode * mode,
            const AFPStation * afp,
            const ProPosPars * prop )
```

Basic Constructor.

sets member variables to corresponding inputs, sets global style, initializes canvas, and builds plot labels

See also

SetROOTStyle(), BuildLabels()

4.11.2.3 ~ProPosCanvas()

```
AFPOpticsStudy::ProPosCanvas::~ProPosCanvas ( )
```

Destructor.

See also

DeleteLabels()

4.11.3 Member Function Documentation

4.11.3.1 Draw()

```
void AFPOpticsStudy::ProPosCanvas::Draw (
            std::vector< FPTracker::Point > points[][2] )
```

draw final proton positions onto canvas with labels

constructs graphs from input points and draws graphs and labels onto canvas

Parameters

points	vector containing FPTracker::Point's that stores the final positions of various example protons

The documentation for this class was generated from the following files:

- AFPOpticsStudy/AFPOpticsStudy/canvases/ProPosCanvas.h
- AFPOpticsStudy/src/canvases/ProPosCanvas.cpp

4.12 AFPOpticsStudy::ProPosPars Class Reference

Storage of parameters used in example proton positions at detector plane.

`#include <ProPosPars.h>`

Public Member Functions

- ProPosPars ()

  Default Constructor.
- ProPosPars (double E_low, double pT_low, double pT_high, int num_angles)

  Basic Constructor.
- double **LowEnergy** () const
- double **LowPT** () const
- double **HighPT** () const
- int **NAngles** () const
- void **SetLowEnergy** (double e)
- void **SetLowPT** (double p)
- void **SetHighPT** (double p)
- void **SetNAngles** (int n)

Private Attributes

- double m_E_low

  [GeV] lower energy to simulate (beam_energy is E_high)
- double m_pT_low

  [GeV] lower transverse momentum to simulate
- double m_pT_high

  [GeV] higher transverse momentum to simulate
- int m_num_angles

  Number of Angles to iterate through (from 0 to twoPI)

4.12.1 Detailed Description

Storage of parameters used in example proton positions at detector plane.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 ProPosPars() [1/2]

```
AFPOpticsStudy::ProPosPars::ProPosPars ( )  [inline]
```

Default Constructor.

Sets member variables to defaults: Low Energy = 6000, Low Transverse Momentum = 0.3, High Transverse Momentum = 0.6, Number of Angles = 20

4.12.2.2 ProPosPars() [2/2]

```
AFPOpticsStudy::ProPosPars::ProPosPars (
            double E_low,
            double pT_low,
            double pT_high,
            int num_angles )  [inline]
```

Basic Constructor.

Sets member variables to corresponding inputs

The documentation for this class was generated from the following file:

- AFPOpticsStudy/AFPOpticsStudy/simulationparameters/ProPosPars.h

4.13 AFPOpticsStudy::Simulator Class Reference

Used to perform simulations and generate random numbers.

```
#include <Simulator.h>
```

Public Member Functions

- Simulator ()

  Default Constructor.
- Simulator (const OpticsMode *mode, const GeoAccPars *gap, const ColAnaPars *cap, const TagTimPars *ttp, const AFPStation *afp)

  Regular Constructor.
- ~Simulator ()

  Default Destructor.
- void Initialize (const OpticsMode *mode, const GeoAccPars *gap, const ColAnaPars *cap, const TagTimPars *ttp, const AFPStation *afp)

  Sets members to inputs.
- void UpdateBeamline ()

Resets the member beamlines to correspond to possible changes in AFPStation or OpticsMode variables.

- bool InAperture (double position) const

  Returns true if position is less than OutOfAperture.
- double Uniform (double min, double max) const

  Returns a random number according to a uniform distribution of [min,max].
- double Poisson (double mean) const

  Returns a random number according to a poisson distribution that has mean.
- double Gaussian (double mean, double stddev) const

  Returns a normally distributed random number with input parameters.
- void Acceptance (TH2D &hist, double pT, double E) const

  Fills hist with the acceptance of the specific transverse momentum and energy.
- std::vector< int > Collimator (std::vector< int > &trials, int c1, int c2) const

  Counts number of particles reaching the detector for given collimator settings.
- std::vector< double > TagTimeProbs (int num_events, double pileup) const

  Calculates probabilities of a particle being tagged and timed.
- FPTracker::Point DetectorPosition (double pz, double pT, double phi) const

  Obtains final position of a particle with input initial properties.
- FPTracker::Point TrajectoryPosition (double obs_pos, double fEl, double px, double py) const

  Obtains final position of a particle with input intial propertries tracked to specific observation position.
- double FloorPathLength (int &situation) const

  Calculates path traveled through the floor of afp roman pot with random intial kinematics.
- std::vector< double > DetectorKinematics (double xIP, double yIP, double zIP, double sxIP, double syIP, double fEl) const

  Calculates particles kinematics at the detector plane.

Private Member Functions

- bool InAFP (double x, double y) const

  Determines if input values are in the AFP rectangle at the detector plane.
- double ColWidth (int c, double beam_width) const

  Calculates the collimator width.
- FPTracker::Beamline ConstructBeamline (double col1, double col2, double det_pos) const

  Constructs beamline using optics settings and inputs.
- FPTracker::Particle ConstructProton (double fracEloss) const

  Constructs particle with specific fractional energy loss and smeared initial kinematics.
- FPTracker::Particle ConstructProtonPolar (double pT, double E) const

  Constructs particle with specific energy and transverse momentum and smeared intial position.

Private Attributes

- const OpticsMode * m_mode

  current mode
- const GeoAccPars * m_gap

  simulation parameters for geometric acceptance
- const ColAnaPars * m_cap

  simulation parameters for collimator analysis
- const TagTimPars * m_ttp

  simulation parameters for tag/time probabilities
- const AFPStation * m_afp

current afp

- FPTracker::Beamline m_beamline

  Beamline that has properties taken from m_mode and m_afp.
- FPTracker::Beamline m_open_beam

  Beamline that has collimators set to open.
- TRandom3 * m_rand_generator

  Utility variable used to generate random numbers.

4.13.1 Detailed Description

Used to perform simulations and generate random numbers.

Uses pointers to the various settings so that simulations can be repeated with different settings without being forced to create a new instance of Simulator. However, the beamlines must be re-constructed if AFPStation distances or collimator settings change. This class is designed to functionally handle all of the simulations dealing with FP↩ Tracker. Almost each method in OpticsStudy has its own specific Simulator function to accomplish its specific goal. In OpticsStudy, there is one instance of Simulator for each afp detector.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 Simulator() [1/2]

```
AFPOpticsStudy::Simulator::Simulator ( )
```

Default Constructor.

Initializes random number generator and that is it

4.13.2.2 Simulator() [2/2]

```
AFPOpticsStudy::Simulator::Simulator (
            const OpticsMode * mode,
            const GeoAccPars * gap,
            const ColAnaPars * cap,
            const TagTimPars * ttp,
            const AFPStation * afp )
```

Regular Constructor.

Sets members to inputs and initializes random number generator

See also

Initialize()

4.13.2.3 ~Simulator()

```
AFPOpticsStudy::Simulator::~Simulator ( )
```

Default Destructor.

Deletes random number generator.

4.13.3 Member Function Documentation

4.13.3.1 Acceptance()

```
void AFPOpticsStudy::Simulator::Acceptance (
            TH2D & hist,
            double pT,
            double E ) const
```

Fills hist with the acceptance of the specific transverse momentum and energy.

Fills histogram with (transverse momentum [GeV] , fractional energy loss , acceptance [%]).

Parameters

hist	two-dimensional histogram that will be filled with acceptance information
pT	specific transverse momentum in GeV
E	specific particle energy in GeV

See also

ConstructProtonPolar()

4.13.3.2 Collimator()

```
std::vector< int > AFPOpticsStudy::Simulator::Collimator (
            std::vector< int > & trials,
            int c1,
            int c2 ) const
```

Counts number of particles reaching the detector for given collimator settings.

The settings of the collimators are found by transforming the input indices via functions defined in ColAnaPars.

Parameters

trials	vector that stores number of trials for each fractional energy loss range (cleared before starting)
c1	index of setting for col1 (TCL4)
c2	index of setting for col2 (TCL5)

Returns

vector that has counted the number of particles hitting the detector for each fractional energy loss range

See also

ColWidth(), ConstructBeamline(), ConstructProton()

4.13.3.3 ColWidth()

```
double AFPOpticsStudy::Simulator::ColWidth (
            int c,
            double beam_width ) const  [private]
```

Calculates the collimator width.

Parameters

c	integer setting as multiple of beam width
beam_width	width of beam at corresponding collimator

Returns

c∗beam_width UNLESS c=0, then returns the open setting for the collimator (999.)

4.13.3.4 ConstructBeamline()

```
FPTracker::Beamline AFPOpticsStudy::Simulator::ConstructBeamline (
            double col1,
            double col2,
            double det_pos ) const  [private]
```

Constructs beamline using optics settings and inputs.

Parameters

col1	setting for col1 width in meters (TCL4)
col2	setting for col2 width in meters (TCL5)
det_pos	position of detector plane in meters

Returns

FPTracker::Beamline that is properly constructed

4.13.3.5 ConstructProton()

```
FPTracker::Particle AFPOpticsStudy::Simulator::ConstructProton (
            double fracEloss ) const  [private]
```

Constructs particle with specific fractional energy loss and smeared initial kinematics.

Smears intial position and momenta according to OpticsMode. Mutliplies z-momentum by negative one if working with beam2. Constructs particle such that it has the input fractional energy loss.

Parameters

fracEloss	fractional energy loss of particle

Returns

FPTracker::Particle that has been constructed

4.13.3.6 ConstructProtonPolar()

```
FPTracker::Particle AFPOpticsStudy::Simulator::ConstructProtonPolar (
            double pT,
            double E ) const  [private]
```

Constructs particle with specific energy and transverse momentum and smeared intial position.

Randomly generates azimuthal angle in uniform distribution from zero to twoPI, and then calculates the particles kinematics from that angle and the inputs.

Parameters

pT	transverse momentum of particle in GeV
E	energy of particle in GeV (used as approximately the z momentum)

Returns

FPTracker::Particle that has been constructed

4.13.3.7 DetectorKinematics()

```
std::vector< double > AFPOpticsStudy::Simulator::DetectorKinematics (
            double xIP,
            double yIP,
            double zIP,
            double sxIP,
```

```
            double syIP,
            double fEl ) const
```

Calculates particles kinematics at the detector plane.

Calculates initial kinematics from inputs and then tracks particle along beamline with open collimators. Includes multiplying zIP and pzIP by -1 if tracking along beam2.

Parameters

xIP	x position at IP in m
yIP	y position at IP in m
zIP	z position at IP in m
sx↩ IP	x inclination angle at IP in rad
sy↩ IP	y inclination angle at IP in rad
fEl	fractional energy loss

Returns

vector storing the kinematics of particle at the detector of the form { x , y , x' , y' }

4.13.3.8 DetectorPosition()

```
FPTracker::Point AFPOpticsStudy::Simulator::DetectorPosition (
            double pz,
            double pT,
            double phi ) const
```

Obtains final position of a particle with input initial properties.

Uses input properties to calculate initial kinematics of particle and then simulates its trajectory through beamline

Parameters

pz	initial z momentum in GeV
pT	initial transverse momentum in GeV
phi	initial azimuthal angle in rad

Returns

FPTracker::Point that contains final position at detector plane relative to beamline

4.13.3.9 FloorPathLength()

```
double AFPOpticsStudy::Simulator::FloorPathLength (
            int & situation ) const
```

Calculates path traveled through the floor of afp roman pot with random intial kinematics.

Explanation of different situations done with image

Parameters

situation	variable storing the integer tag value for a given situation

Returns

path length traveled through floor in m

4.13.3.10 InAFP()

```
bool AFPOpticsStudy::Simulator::InAFP (
            double x,
            double y ) const  [private]
```

Determines if input values are in the AFP rectangle at the detector plane.

Assumes AFPStation is on the negative x-axis and uses values stored in m_afp.

Parameters

x	x position in mm
y	y poisiton in mm

Returns

true if (x,y) is in box defined by m_afp, false otherwise

4.13.3.11 Initialize()

```
void AFPOpticsStudy::Simulator::Initialize (
            const OpticsMode * mode,
            const GeoAccPars * gap,
            const ColAnaPars * cap,
            const TagTimPars * ttp,
            const AFPStation * afp )
```

Sets members to inputs.

Sets the member variables to the settings corresponding to the inputs Overrides previous settings if there are any.

Parameters

mode	pointer to OpticsMode used in simulation
gap	pointer to GeoAccPars used in simulation
cap	pointer to ColAnaPars used in simulation
ttp	pointer to TagTimPars used in simulation
afp	pointer to AFPStation used in simulation

See also

UpdateBeamline()

4.13.3.12 TagTimeProbs()

```
std::vector< double > AFPOpticsStudy::Simulator::TagTimeProbs (
            int num_events,
            double pileup ) const
```

Calculates probabilities of a particle being tagged and timed.

Uses single and double tag probabilities from OpticsMode to obtain probabilities of tagging and timing events for a given amount of pileup. Obtains time of flight settings for AFP from TagTimPars member.

Parameters

num_events	number of events to simulate
pileup	value of pileup at interaction point

Returns

vector of probabilities of the form { tagging prob , timing prob 1 , timing prob 2 , ... }

4.13.3.13 TrajectoryPosition()

```
FPTracker::Point AFPOpticsStudy::Simulator::TrajectoryPosition (
            double obs_pos,
            double fEl,
            double px,
            double py ) const
```

Obtains final position of a particle with input intial propertries tracked to specific observation position.

Constructs beamline corresponding to observation postion

Parameters

obs_pos	observation position in m
fEl	fractional energy loss of particle
px	x momentum of particle
py	y momentum of particle

Returns

FPTracker::Point that contains the position at obs_pos relative to beamline

See also

ConstructBeamline()

4.13.3.14 UpdateBeamline()

```
void AFPOpticsStudy::Simulator::UpdateBeamline ( )
```

Resets the member beamlines to correspond to possible changes in AFPStation or OpticsMode variables.

Pulls necessary variables from member pointers to AFPStation and OpticsMode.

See also

ConstructBeamline()

The documentation for this class was generated from the following files:

- AFPOpticsStudy/AFPOpticsStudy/Simulator.h
- AFPOpticsStudy/src/Simulator.cpp
- AFPOpticsStudy/src/Simulator_Utility.cpp

4.14 AFPOpticsStudy::SpecificParam Class Reference

Storage and evaluation of parameterisation with specific degrees.

```
#include <SpecificParam.h>
```

Public Member Functions

- SpecificParam ()

  Default Constructor.
- SpecificParam (const std::string &name, const std::vector< std::vector< std::vector< double > > > &coefficients)

  General Constructor.
- void **SetName** (const std::string &name)
- std::string **Name** () const
- void Load (const std::string &filepath)

  copies coefficients into member variable and determines maximum degree
- void PrintFunc (std::ostream &out) const

  Prints string form of functions that will be evaluated to out.
- void Print (std::ostream &out) const

  Prints coefficients table to out (in the traditional style)
- std::vector< double > Eval (const double xip, const double yip, const double zip, const double sxip, const double syip, const double fEl) const

  Returns a vector that contains the detector kinematics according to SpecificParam.

Private Member Functions

- void SetMaximumDegree ()

  Finds maximum degree from sizes of vectors in coefficient table and sets member variable to it (sets to 0 if coefficient table is not filled)

Private Attributes

- std::string m_name

  name of parameterisation (printed on histograms and files)
- std::vector< std::vector< std::vector< double > > > m_coefficients
- bool m_coefficients_filled

  true if m_coefficients has been set
- int m_maximum_degree

  stores maximum degree in polynomial

4.14.1 Detailed Description

Storage and evaluation of parameterisation with specific degrees.

This is a helper class so that the Parameterisation class would not have to store as much information. Each SpecificParam corresponds to a transport parameterisation that has a chosen degree for each of the member polynomials. A name for the parameterisation is also included so that other classes can name the files and histograms automatically and multiple different transport parameterisations can be investigated from the same Parameterisation class. This class can also handle reading in the parameterisation from a file. Currently, the class ignores the header portion of the parameterisation file, choosing to keep the values set by the user; however, it would be simple to incorporate the reading of this header portion into member variables of this class or into the member variables of other classes (AFPStation and OpticsMode).

4.14.2 Constructor & Destructor Documentation

4.14.2.1 SpecificParam() [1/2]

```
AFPOpticsStudy::SpecificParam::SpecificParam ( )
```

Default Constructor.

Sets check member variable to false

4.14.2.2 SpecificParam() [2/2]

```
AFPOpticsStudy::SpecificParam::SpecificParam (
            const std::string & name,
            const std::vector< std::vector< std::vector< double > > > & coefficients )
```

General Constructor.

Copies over input coefficient table and determines maximum degree

See also

SetMaximumDegree()

4.14.3 Member Function Documentation

4.14.3.1 Eval()

```
std::vector< double > AFPOpticsStudy::SpecificParam::Eval (
            const double xip,
            const double yip,
            const double zip,
            const double sxip,
            const double syip,
            const double fEl ) const
```

Returns a vector that contains the detector kinematics according to SpecificParam.

Parameters

xip	x position at IP
yip	y position at IP
zip	z position at IP
sxip	sx = px/pz at IP
syip	sy = py/pz at IP
fEl	fractional energy loss

Returns

detector kinematics in vector: { x , y , sx , sy }

4.14.3.2 Load()

```
void AFPOpticsStudy::SpecificParam::Load (
            const std::string & filepath )
```

copies coefficients into member variable and determines maximum degree

Replaces whatever coefficients were there Sets name to name of file (everything between last / and extension .txt)

See also

SetMaximumDegree()

4.14.3.3 Print()

```
void AFPOpticsStudy::SpecificParam::Print (
            std::ostream & out ) const
```

Prints coefficients table to out (in the traditional style)

Traditional style for coefficients of transport parameterisation: Each function is separated by a single '@' on a line by itself. After the '@', the polynomials are listed from A to H where the degree is listed first and then all of the coefficients are listed on the same line separated by spaces. The header of the parameterisation file is NOT printed using this function (see OpticsStudy::PrintParameterisationHeader()).

4.14.3.4 SetMaximumDegree()

```
void AFPOpticsStudy::SpecificParam::SetMaximumDegree ( )  [private]
```

Finds maximum degree from sizes of vectors in coefficient table and sets member variable to it (sets to 0 if coefficient table is not filled)

Knowing the maximum degree in fractional energy loss is useful for the Eval() function.

4.14.4 Member Data Documentation

4.14.4.1 m_coefficients

```
std::vector< std::vector< std::vector<double> > > AFPOpticsStudy::SpecificParam::m_coefficients
[private]
```

Values of coefficients for a specific parameterisation 1st : which observable 2nd : which polynomial 3rd : index of coefficient

The documentation for this class was generated from the following files:

- AFPOpticsStudy/AFPOpticsStudy/parameterisation/SpecificParam.h
- AFPOpticsStudy/src/parameterisation/SpecificParam.cxx

4.15 AFPOpticsStudy::TagTimPars Class Reference

Storage of parameters used in calculation of tagging and timing probabilities.

```
#include <TagTimPars.h>
```

Public Member Functions

- TagTimPars ()
  Default Constructor.
- TagTimPars (double pileupMIN, double pileupMAX, std::vector< double > ToF_settings)
  Basic Constructor.
- double **MinPileup** () const
- double **MaxPileup** () const
- double **ToFSettings** (int i) const
- int **NProbs** () const
- double **EventsMultiplier** () const
- void **SetMinPileup** (double p)
- void **SetMaxPileup** (double p)
- void **SetToFSettings** (std::vector< double > v)
- void **SetEventsMultiplier** (double n)

Private Attributes

- double m_pileupMIN
  minimum pileup to simulate for
- double m_pileupMAX
  maximum pileup to simulate for
- double m_events_multiplier
  Scales the number of events.
- std::vector< double > m_ToF_settings
  vector corresponding to different time of flight settings for the AFP
- int m_num_probs
  Number of different probabilities calculated for each pileup value (m_ToF_settings.size()+2)

4.15.1 Detailed Description

Storage of parameters used in calculation of tagging and timing probabilities.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 TagTimPars() [1/2]

```
AFPOpticsStudy::TagTimPars::TagTimPars ( )  [inline]
```

Default Constructor.

Sets member variables to defaults: maximum pileup = 80.001, time of flight settings = { 6 }, Number of Probabilities = 3, Events Multiplier = 1,

4.15.2.2 TagTimPars() [2/2]

```
AFPOpticsStudy::TagTimPars::TagTimPars (
            double pileupMIN,
            double pileupMAX,
            std::vector< double > ToF_settings )  [inline]
```

Basic Constructor.

Sets member variables to corresponding inputs and calculates number of probabilities. Adds 0.001 to input maximum pileup, so that it will more likely be reached by while loop.

The documentation for this class was generated from the following file:

- AFPOpticsStudy/AFPOpticsStudy/simulationparameters/TagTimPars.h

4.16 AFPOpticsStudy::TrajExCanvas Class Reference

drawing and printing of example trajectories

```
#include <TrajExCanvas.h>
```

Public Member Functions

- TrajExCanvas ()
  Default Constructor.
- TrajExCanvas (const OpticsMode *mode, const AFPStation *afp_near, const AFPStation *afp_far, const TrajExPars *tep, bool drawAPER)
  Basic Constructor.
- ~TrajExCanvas ()
  Destructor.
- void Draw (std::vector< FPTracker::Point > points[ ], int q, int e)
  draws specific trajectories onto canvas along with corresponding labels/graphs
- void Save (std::string filepath)
  saves canvas to filepath

Private Member Functions

- void SetROOTStyle ()
  set global printing and drawing style
- void BuildLabels ()
  build labels and graphs from current mode and afp stations
- void DeleteLabels ()
  delete labels and graphss
- void DrawLabels (int xory, int leg)
  draw labels and graphs (not trajectories)

Private Attributes

- const OpticsMode * m_mode
  current mode
- const AFPStation * m_afp_near
  current near afp station
- const AFPStation * m_afp_far
  current far afp station
- const TrajExPars * m_tep
  example trajectory simulation parameters
- TCanvas * m_trajexcanvas
  drawing and printing canvas
- bool m_draw_aperture
  whether or not to draw aperture
- TGraph * m_frame [2]
  frames for both x and y directions (0<=>x , 1<=>y)
- TGraph * m_aperture [2][2]
  graphs to draw aperture ({x or y}, {+/- side})
- std::vector< TGraph * > m_gr [2][3]
  graphs of trajectories ({x or y} , {fEl, px, or py})
- TLegend * m_legend [2][3]
  legends for different trajectories ({x or y}, {fEl, px, or py})
- std::vector< TBox * > m_mags [2]
  plots of magnets for both x and y directions
- std::vector< TLatex * > m_maglabs [2]
  labels of magnets for both x and y directions
- std::vector< TLine * > m_lines [2]
  plots of lines for both x and y directions
- std::vector< TLatex * > m_linelabs [2]
  labels of lines for both x and y directions
- TLatex * m_vertex [2]
  labels the vertex of the simulation {x or y}
- TLatex * m_nonchanging [2][3]
  labels kinematics that aren't in the legend ({x or y}, {fEl, px, or py})

4.16.1 Detailed Description

drawing and printing of example trajectories

4.16.2 Constructor & Destructor Documentation

4.16.2.1 TrajExCanvas() [1/2]

```
AFPOpticsStudy::TrajExCanvas::TrajExCanvas ( )
```

Default Constructor.

sets global style and intializes canvas

See also

SetROOTStyle()

4.16.2.2 TrajExCanvas() [2/2]

```
AFPOpticsStudy::TrajExCanvas::TrajExCanvas (
            const OpticsMode * mode,
            const AFPStation * afp_near,
            const AFPStation * afp_far,
            const TrajExPars * tep,
            bool drawAPER )
```

Basic Constructor.

sets member variables to corresponding inputs, sets global style, initializes canvas, and builds plot labels/graphs

See also

SetROOTStyle(), BuildLabels()

4.16.2.3 ~TrajExCanvas()

```
AFPOpticsStudy::TrajExCanvas::~TrajExCanvas ( )
```

Destructor.

See also

DeleteLabels()

4.16.3 Member Function Documentation

4.16.3.1 Draw()

```
void AFPOpticsStudy::TrajExCanvas::Draw (
            std::vector< FPTracker::Point > points[],
            int q,
            int e )
```

draws specific trajectories onto canvas along with corresponding labels/graphs

Parameters

q	which axis to draw (0<=>x,1<=>y)
e	which legend to draw (0<=>fEl , 1<=>px , 2<=>py)
points	contains vectors of FPTracker::Point s that are the trajectories to be plotted

See also

DrawLabels()

4.16.3.2 DrawLabels()

```
void AFPOpticsStudy::TrajExCanvas::DrawLabels (
            int xory,
            int leg )  [private]
```

draw labels and graphs (not trajectories)

Parameters

xory	which axis to draw (0<=>x,1<=>y)
leg	which legend to draw (0<=>legendfEl , 1<=>legendpx , 2<=>legendpy)

The documentation for this class was generated from the following files:

- AFPOpticsStudy/AFPOpticsStudy/canvases/TrajExCanvas.h
- AFPOpticsStudy/src/canvases/TrajExCanvas.cpp

4.17 AFPOpticsStudy::TrajExPars Class Reference

Storage of parameters for simulating example trajectories of the proton.

```
#include <TrajExPars.h>
```

Public Member Functions

- TrajExPars ()

  Default Constructor.
- TrajExPars (double obs_posSTEP, double obs_posMAX, std::vector< double > fEl, std::vector< double > pT)

  Basic Constructor.
- double **StepObservePosition** () const
- double **MaxObservePosition** () const
- int **NFracELossOpts** () const

- int **NTransversePOpts** () const
- double **FracELossOpt** (int i) const
- double **TransversePOpt** (int i) const
- void **SetStepObservePosition** (double s)
- void **SetMaxObservePosition** (double s)
- void **SetFracELossOpts** (std::vector< double > v)
- void **SetTransversePOpts** (std::vector< double > v)

Private Attributes

- double m_obs_posSTEP

  [m] distance between successive steps of the observation postion
- double m_obs_posMAX

  [m] maximum observation postion
- std::vector< double > m_fEl

  vector of different fractional energy losses to simulate
- std::vector< double > m_pT

  [GeV] vector of different transverse momenta to simulate (used for varying both px and py)

4.17.1 Detailed Description

Storage of parameters for simulating example trajectories of the proton.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 TrajExPars() [1/2]

```
AFPOpticsStudy::TrajExPars::TrajExPars ( )  [inline]
```

Default Constructor.

Sets member variables to defaults: observation position step = 1 maximum observation position = 250 fraction energy loss options = { 0 , 0.04 , 0.08 , 0.12 , 0.16 } transverse momenta options = { 1. , 0.5 , 0. , -0.5 , -1. }

4.17.2.2 TrajExPars() [2/2]

```
AFPOpticsStudy::TrajExPars::TrajExPars (
            double obs_posSTEP,
            double obs_posMAX,
            std::vector< double > fEl,
            std::vector< double > pT )  [inline]
```

Basic Constructor.

Sets member variables to corresponding inputs

The documentation for this class was generated from the following file:

- AFPOpticsStudy/AFPOpticsStudy/simulationparameters/TrajExPars.h

4.18 AFPOpticsStudy::WriteTwiss Class Reference

Class Defined to write necessary twiss files using MAD-X program.

`#include <WriteTwiss.h>`

Public Member Functions

- WriteTwiss (const std::string b1_seq, const std::vector< std::string > b1_m_i, const std::string b2_seq, const std::vector< std::string > b2_m_i, const double beam_energy, const double crossing_angle, const double normalized_emittance, const std::string product_dir, const AFPStation *a_far, const AFPStation *a_near, const AFPStation *c_near, const AFPStation *c_far, const std::string discription)

  Writing Constructor.
- void WriteTwissFile (bool isb2)

  Uses madx program to write twiss file into twiss directory.
- std::string **TwissDir** ()
- std::string **ParentDir** ()

Private Member Functions

- bool BuildMADXFile (bool isb2, std::string &madx_filepath)

  Constructs madx file for beam depending on inputs and sets madx_filepath to constructed file.

Private Attributes

- const int s_iA_FAR = 0

  Index used internally for afp station A FAR.
- const int s_iA_NEAR = 1

  Index used internally for afp station A NEAR.
- const int s_iC_NEAR = 2

  Index used internally for afp station C NEAR.
- const int s_iC_FAR = 3

  Index used internally for afp station C FAR.
- const std::string m_beam1_madx_sequence

  filepath to madx sequence file for beam1 (defined by user)
- const std::vector< std::string > m_beam1_madx_in

  filepaths to other madx files required (defined by user)
- const std::string m_beam2_madx_sequence

  filepath to madx sequence file for beam2 (defined by user)
- const std::vector< std::string > m_beam2_madx_in

  filepaths to other madx files required (defined by user)
- const double m_beam_energy

  Energy of Beam (defined by user) [GeV].
- const double m_crossing_angle

  Crossing Angle at IP (defined by user) [murad].
- const double m_normalized_emittance

  *Normalized emittance at IP (defined by user) [m*rad].*
- std::vector< const AFPStation * > m_afp

  Pointers to the four afp stations (defined by user)

- const std::string m_product_dir

  Filepath to directory in which files should be stored (defined by user)
- const std::string m_description

  string added to name of parent directory (after beam energy and crossing angle)
- std::string m_parent_dir

  Directory in m_product_dir created to store files.
- std::string m_twiss_dir

  Directory in m_parent_dir created to store twiss files.
- std::string m_madx_dir

  Directory in m_parent_dir created to store madx files.
- std::string m_madx_beam1_file

  filepath that goes to the madx file to create twiss file for beam1
- std::string m_madx_beam2_file

  filepath that goes to the madx file to create twiss file for beam2a
- boost::system::error_code m_retERR

  utility variable to check if directories were created successfully
- bool m_made_beta_plot

  utility variable to save whether or not the beta function plot has already been made

4.18.1 Detailed Description

Class Defined to write necessary twiss files using MAD-X program.

This class is used functionally and not for storage. It is not used as member of the interface class OpticsStudy because it is only called in the MAD-X constructor of OpticsStudy. The thought is that after the program creates the necessary twiss files, those are all that is needed to run the simulations using FPTracker. This class also creates the necessary directories from from the input beam properties (namely beam energy and crossing angle).

4.18.2 Constructor & Destructor Documentation

4.18.2.1 WriteTwiss()

```
AFPOpticsStudy::WriteTwiss::WriteTwiss (
            const std::string b1_seq,
            const std::vector< std::string > b1_m_i,
            const std::string b2_seq,
            const std::vector< std::string > b2_m_i,
            const double beam_energy,
            const double crossing_angle,
            const double normalized_emittance,
            const std::string product_dir,
            const AFPStation * a_far,
            const AFPStation * a_near,
            const AFPStation * c_near,
            const AFPStation * c_far,
            const std::string discription )
```

Writing Constructor.

Takes inputs and sets corresponding member variables (including constructing necessary directories)

Parameters

b1_seq	filepath to madx sequence file for beam1
b1_m_i	vector of filepaths to other required madx files for beam1
b2_seq	filepath to madx sequence file for beam2
b2_m_i	vector of filepaths to other required madx files for beam2
beam_energy	energy of beam in GeV
crossing_angle	crossing angle in s-y plane in murad
normalized_emittance	normalized emittance at IP in m∗rad
product_dir	filepath to directory in which all directories and files will be stored
a_far	pointer to AFPStation A FAR
a_near	pointer to AFPStation A NEAR
c_near	pointer to AFPStation C NEAR
c_far	pointer to AFPStation C FAR

4.18.3 Member Function Documentation

4.18.3.1 BuildMADXFile()

```
bool AFPOpticsStudy::WriteTwiss::BuildMADXFile (
            bool isb2,
            std::string & madx_filepath )  [private]
```

Constructs madx file for beam depending on inputs and sets madx_filepath to constructed file.

Writes madx file as an ofstream. Sets member variable m_madx_beam1_file or m_madx_beam2_file depending on input.

Parameters

isb2	defining which beam (true <=> beam2 , false <=> beam1)
madx_filepath	variable that is set to constructed file path

Returns

true if successfully wrote madx file (false otherwise)

4.18.3.2 WriteTwissFile()

```
void AFPOpticsStudy::WriteTwiss::WriteTwissFile (
            bool isb2 )
```

Uses madx program to write twiss file into twiss directory.

Takes input beam and writes madx file and then uses written madx file to steer madx program into writing twiss file.

Parameters

isb2	defining which beam (true <=> beam2 , false <=> beam1)

See also

BuildMADXFile()

The documentation for this class was generated from the following files:

- AFPOpticsStudy/AFPOpticsStudy/WriteTwiss.h
- AFPOpticsStudy/src/WriteTwiss.cpp

Index