PPF C++ Reference Manual

Generated by Doxygen 1.4.6-NO

Fri Mar 16 17:18:01 2007

Contents

Chapter 1

PPF C++ Namespace Index

1.1 PPF C++ Namespace List

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

Chapter 2

PPF C++ Hierarchical Index

2.1 PPF C++ Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

PPF C++ Class Index

3.1 PPF C++ Class List

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

Chapter 4

PPF C++ Namespace Documentation

4.1 ppf Namespace Reference

Contains the PPF c++ wrapper classes and types.

Classes

- class **datatype**

  Contains signal data requested from the PPF system.

- class **system**

  The main PPF system class through which all PPF access is performed.

- class **coord**

  A concise container for an (x,t) co-ordinate and corresponding data value.

- class **ppf_error**

  PPF system exception class.

Typedefs

- typedef int **accessflag**

  Access flags are used in conjunction with the connect() method to specify read and write access to the PPF system.

- typedef int **ioflag**

  I/O flags can be used to modify the reading and writing of PPF DDA datatypes.

Enumerations

- enum **processflag** { **nearest**, **linear**, **preserve** }

  Process flags can be used to modify the operation of a number of data get and set methods.

Variables

- **system ppfdb**

  PPF database access object.

- **system ppfdb**

  PPF database access object.

- static const **accessflag read** = 0x01

  Open PPF system for reading.

- static const **accessflag write** = 0x02

  Open PPF system for writing.

- static const **ioflag tvector** = 0x01

  Enables reading or writing of the T vector.

- static const **ioflag xvector** = 0x02

  Enables reading or writing of the X vector.

- static const **ioflag noxvector** = 0x04

  Disables reading or writing of the X vector.

- static const **ioflag notvector** = 0x08

  Disables reading or writing of the T vector.

- static const **ioflag lastxvector** = 0x10

  (Writing only) Instructs the PPF system to reuse the X vector of a previously written datatype.

- static const **ioflag lasttvector** = 0x20

  (Writing only) Instructs the PPF system to reuse the T vector of a previously written datatype.

4.1.1 Detailed Description

Contains the PPF c++ wrapper classes and types.

4.1.2 Enumeration Type Documentation

4.1.2.1 enum ppf::processflag

Process flags can be used to modify the operation of a number of data get and set methods.

Enumerator:

nearest Use nearest indexed value.

linear Use linear interpolation.

preserve No not modify data.

Chapter 5

PPF C++ Class Documentation

5.1 ppf::coord Class Reference

A concise container for an (x,t) co-ordinate and corresponding data value.

`#include <ppftypes.hpp>`

Public Attributes

- float **t**
  t coordinate
- float **x**
  x coordinate
- float **d**
  *value at(*t, x*)*

5.1.1 Detailed Description

A concise container for an (x,t) co-ordinate and corresponding data value.

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

- Q:/Codes/PPF/FullPPF/ppftypes.hpp

5.2 ppf::datatype Class Reference

Contains signal data requested from the PPF system.

`#include <ppfdatatype.hpp>`

Public Member Functions

- **datatype** ()

  Constructor.

- **datatype** (const string filename)

  Initialising constructor.

- void **resizeT** (const int nt)

  Resizes the T and data vectors to the specified number of elements along the T axis.

- void **resizeX** (const int nx)

  Resizes the X and data vectors to the specified number of elements along the X axis.

- void **resizeTX** (int nt, int nx)

  Resizes the T, X and data vectors to the specified number of elements along the T and X axes.

- void **clear** ()

  Clears the datatype by setting it back to its default values.

- bool **empty** () const

  Returns true if the T and X dimensions of the datatype are zero, false otherwise.

- void **readLocal** (const string filename)

  Reads in data from a local file.

- void **writeLocal** (const string filename)

  Reads in data from a local file.

- int **sizeT** () const

  Returns the T dimension of the data and T vectors.

- int **sizeX** () const

  Returns the X dimension of the data and X vectors.

- string **getTUnit** () const

  Returns the units of the T vector.

- string **getXUnit** () const

Returns the units of the X vector.

- string **getDataUnit** () const

  Returns the units of the data vector.

- string **getComment** () const

  Returns the datatype comment.

- int **getUserStatus** () const

  Returns the user status of the datatype.

- int **getSysStatus** () const

  Returns the system status of the datatype.

- float **getT** (const int It) const

  Returns the T vector value at index It.

- float **getX** (const int Ix) const

  Returns the X vector value at index Ix.

- float **getDataIt** (const int It) const

  Returns the data value at index It *(*Ix=0*).*

- float **getDataIx** (const int Ix) const

  Returns the data value at index Ix *(*It=0*).*

- float **getDataItIx** (const int It, const int Ix) const

  *Returns the data value at index [*It, Ix*].*

- int **getIt** (const float t) const

  Returns the closest T vector index to the passed time t.

- int **getIx** (const float x) const

  Returns the closest X vector index to the passed x *co-ordinate.*

- float **getDataT** (const float t) const

  Returns linearly interpolated data value at time t *(assumes* Ix=0*).*

- float **getDataT** (const float t, const **processflag** flag) const

  Returns the data value at time t *using the method specified by the passed process flag (assumes* Ix=0*).*

- **coord getDataCT** (const float t, const **processflag** flag) const

  Returns a packed coordinate (t, x, data value) at time t *using the method specified by the passed process flag (assumes* Ix=0*).*

- float **getDataX** (const float x) const

Returns linearly interpolated data value at x *(assumes* It=0*).*

- float **getDataX** (const float x, const **processflag** flag) const

  Returns the data value at x *using the method specified by the passed process flag (assumes* It=0*).*

- **coord getDataCX** (const float x, const **processflag** flag) const

  Returns a packed coordinate (t, x, data value) at x *using the method specified by the passed process flag (assumes* It=0*).*

- float **getDataTIx** (const float t, const long Ix) const

  Returns linearly interpolated data value at time t *and X index* Ix*.*

- float **getDataTIx** (const float t, const long Ix, const **processflag** t_flag) const

  Returns the data value at time t *and X index* Ix *using the method specified by the passed process flag.*

- **coord getDataCTIx** (const float t, const long Ix, const **processflag** t_flag) const

  Returns a packed coordinate (t, x, data value) at time t *and X index* Ix *using the method specified by the passed process flag.*

- float **getDataItX** (const long It, const float x) const

  Returns linearly interpolated data value at x *and T index* It*.*

- float **getDataItX** (const long It, const float x, const **processflag** x_flag) const

  Returns the data value at x *and T index* It *using the method specified by the passed process flag.*

- **coord getDataCItX** (const long It, const float x, const **processflag** x_flag) const

  Returns a packed coordinate (t, x, data value) at x *and T index* It *using the method specified by the passed process flag (assumes* It=0*).*

- float **getDataTX** (const float t, const float x) const

  Returns linearly interpolated data value at time t *and co-ordinate* x*.*

- float **getDataTX** (const float t, const float x, const **processflag** t_flag, const **processflag** x_flag) const

  Returns the data value at time t *and co-ordinate* x *using the method specified by the passed process flag.*

- **coord getDataCTX** (const float t, const float x, const **processflag** t_flag, const **processflag** x_flag) const

  Returns a packed coordinate (t, x, data value) at time t *and co-ordinate* x *using the method specified by the passed process flag.*

- void **setTUnit** (const string s)

Sets the units of the T vector.

- void **setXUnit** (const string s)

  Sets the units of the X vector The maximum string length is 8 chars, characters beyond this limit will be removed.

- void **setDataUnit** (const string s)

  Sets the units of the data vector The maximum string length is 8 chars, characters beyond this limit will be removed.

- void **setComment** (const string s)

  Sets the datatype comment The maximum string length is 24 chars, characters beyond this limit will be removed.

- void **setUserStatus** (const int v)

  Sets the user status of the datatype.

- void **setSysStatus** (const int v)

  Sets the system status of the datatype.

- void **setT** (const int It, const float v)

  Sets the T vector value at index It*.*

- void **setX** (const int Ix, const float v)

  Sets the X vector value at index Ix*.*

- void **setDataIt** (const int It, const float v)

  Sets the data value at index It *(*Ix=0*).*

- void **setDataIx** (const int Ix, const float v)

  Sets the data value at index Ix *(*It=0*).*

- void **setDataItIx** (const int It, const int Ix, const float v)

  *Sets the data value at index [*It*,* Ix*].*

5.2.1 Detailed Description

Contains signal data requested from the PPF system.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 ppf::datatype::datatype (const string *filename*) [inline]

Initialising constructor.

Reads in data from a local file.

Parameters:
filename String containing the filename (including path)

5.2.3 Member Function Documentation

5.2.3.1 void ppf::datatype::clear ()

Clears the datatype by setting it back to its default values.

All contents of the datatype will be erased including comments, units and status flags.

5.2.3.2 bool ppf::datatype::empty () const [inline]

Returns true if the T and X dimensions of the datatype are zero, false otherwise.

Meta-data such as the comments, units and status flags are not checked by a call to **empty**()(p. 16).

5.2.3.3 string ppf::datatype::getComment () const [inline]

Returns the datatype comment.

Returns:
A string containing the datatype comment

5.2.3.4 coord ppf::datatype::getDataCItX (const long *It*, const float *x*, const processflag *x_flag*) const

Returns a packed coordinate (t, x, data value) at *x* and T index *It* using the method specified by the passed process flag (assumes *It=0*).

Valid process flags are: *nearest*, *linear*, *bicubic* This routine will only be successful provided the X vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:
x x co-ordinate

It index in T-direction

x_flag a valid process flag

Returns:
A packed coordinate containing the exact t, x and data values

5.2.3.5 coord ppf::datatype::getDataCT (const float *t*, const processflag *flag*) const

Returns a packed coordinate (t, x, data value) at time *t* using the method specified by the passed process flag (assumes *Ix=0*).

Valid process flags are: *nearest*, *linear*, *bicubic* This routine will only be successful provided the T vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:

t time

flag a valid process flag

Returns:

A packed coordinate containing the exact *t*, *x* and *data* values

5.2.3.6 coord ppf::datatype::getDataCTIx (const float *t*, const long *Ix*, const processflag *t_flag*) const

Returns a packed coordinate (t, x, data value) at time *t* and X index *Ix* using the method specified by the passed process flag.

Valid process flags are: *nearest*, *linear*, *bicubic* This routine will only be successful provided the T vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:

t time

Ix index in X-direction

t_flag a valid process flag

Returns:

A packed coordinate containing the exact *t*, *x* and *data* values

5.2.3.7 coord ppf::datatype::getDataCTX (const float *t*, const float *x*, const processflag *t_flag*, const processflag *x_flag*) const

Returns a packed coordinate (t, x, data value) at time *t* and co-ordinate *x* using the method specified by the passed process flag.

Valid process flags are: *nearest*, *linear*, *bicubic* This routine will only be successful provided the T and X vectors are monotonically increasing or decreasing and no two values are equal in each vector, its result is undefined if applied to data not satisfying these conditions.

Parameters:

t time

x x co-ordinate

t_flag a valid process flag for the T vector

x_flag a valid process flag for the X vector

Returns:
A packed coordinate containing the exact *t*, *x* and *data* values

5.2.3.8 coord ppf::datatype::getDataCX (const float *x*, const processflag *flag*) const

Returns a packed coordinate (t, x, data value) at *x* using the method specified by the passed process flag (assumes *It=0*).

Valid process flags are: *nearest*, *linear*, *bicubic* This routine will only be successful provided the X vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:
x x co-ordinate

flag a valid process flag

Returns:
A packed coordinate containing the exact t, x and data values

5.2.3.9 float ppf::datatype::getDataIt (const int *It*) const [inline]

Returns the data value at index *It* (*Ix=0*).

If the ppf is 2D then the X index is always taken to be zero.

Parameters:
It index in T-direction

Returns:
The data value at index *It*

5.2.3.10 float ppf::datatype::getDataItIx (const int *It*, const int *Ix*) const [inline]

Returns the data value at index [*It*, *Ix*].

Parameters:
It index in T-direction

Ix index in X-direction

Returns:
The data value at index [*It*, *Ix*].

5.2.3.11 float ppf::datatype::getDataItX (const long *It*, const float *x*, const processflag *x_flag*) const

Returns the data value at *x* and T index *It* using the method specified by the passed process flag.

Valid process flags are: *nearest*, *linear*, *bicubic* This routine will only be successful provided the X vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:

x x co-ordinate

It index in T-direction

x_flag a valid process flag

Returns:

The data value

5.2.3.12 float ppf::datatype::getDataItX (const long *It*, const float *x*) const [inline]

Returns linearly interpolated data value at *x* and T index *It*.

This routine will only be successful provided the X vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:

x x co-ordinate

It index in T-direction

Returns:

The linearly interpolated data value at *x* and index *It*

5.2.3.13 float ppf::datatype::getDataIx (const int *Ix*) const [inline]

Returns the data value at index *Ix* (*It=0*).

If the ppf is 2D then the T index is always taken to be zero.

Parameters:

Ix index in X-direction

Returns:

The data value at index *Ix*

5.2.3.14 float ppf::datatype::getDataT (const float *t*, const processflag *flag*) const

Returns the data value at time *t* using the method specified by the passed process flag (assumes *Ix=0*).

Valid process flags are: *nearest*, *linear*, *bicubic* This routine will only be successful provided the T vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:

t time

flag a valid process flag

Returns:

The data value

5.2.3.15 float ppf::datatype::getDataT (const float *t*) const [inline]

Returns linearly interpolated data value at time *t* (assumes *Ix=0*).

This routine will only be successful provided the T vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:

t time

Returns:

The linearly interpolated data value at time *t*

5.2.3.16 float ppf::datatype::getDataTIx (const float *t*, const long *Ix*, const processflag *t_flag*) const

Returns the data value at time *t* and X index *Ix* using the method specified by the passed process flag.

Valid process flags are: *nearest*, *linear*, *bicubic* This routine will only be successful provided the T vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:

t time

Ix index in X-direction

t_flag a valid process flag

Returns:

The data value

5.2.3.17 float ppf::datatype::getDataTIx (const float *t*, const long *Ix*) const [inline]

Returns linearly interpolated data value at time *t* and X index *Ix*.

This routine will only be successful provided the T vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:

t time

Ix index in X-direction

Returns:

The linearly interpolated data value at time *t* and index *Ix*

5.2.3.18 float ppf::datatype::getDataTX (const float *t*, const float *x*, const processflag *t_flag*, const processflag *x_flag*) const

Returns the data value at time *t* and co-ordinate *x* using the method specified by the passed process flag.

Valid process flags are: *nearest*, *linear*, *bicubic* This routine will only be successful provided the T and X vectors are monotonically increasing or decreasing and no two values are equal in each vector, its result is undefined if applied to data not satisfying these conditions.

Parameters:

t time

x x co-ordinate

t_flag a valid process flag for the T vector

x_flag a valid process flag for the X vector

Returns:

The data value

5.2.3.19 float ppf::datatype::getDataTX (const float *t*, const float *x*) const [inline]

Returns linearly interpolated data value at time *t* and co-ordinate *x*.

This routine will only be successful provided the T and X vectors are monotonically increasing or decreasing and no two values are equal in each vector, its result is undefined if applied to data not satisfying these conditions.

Parameters:

t time

x x co-ordinate

Returns:
The linearly interpolated data value at time *t* and co-ordinate *x*

5.2.3.20 string ppf::datatype::getDataUnit () const [inline]

Returns the units of the data vector.

Returns:
A string containing the units of the data vector

5.2.3.21 float ppf::datatype::getDataX (const float *x*, const processflag *flag*) const

Returns the data value at *x* using the method specified by the passed process flag (assumes *It=0*).

Valid process flags are: *nearest*, *linear*, *bicubic* This routine will only be successful provided the X vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:
x x co-ordinate

flag a valid process flag

Returns:
The data value

5.2.3.22 float ppf::datatype::getDataX (const float *x*) const [inline]

Returns linearly interpolated data value at *x* (assumes *It=0*).

This routine will only be successful provided the X vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:
x x co-ordinate

Returns:
The linearly interpolated data value at *x*

5.2.3.23 int ppf::datatype::getIt (const float *t*) const

Returns the closest T vector index to the passed time *t*.

This routine will only be successful provided the T vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:
t time

Returns:
The T vector index closest to passed time *t*

5.2.3.24 int ppf::datatype::getIx (const float *x*) const

Returns the closest X vector index to the passed *x* co-ordinate.

This routine will only be successful provided the X vector is monotonically increasing or decreasing and no two values are equal, its result is undefined if applied to data not satisfying these conditions.

Parameters:
x co-ordinate

Returns:
The X vector index closest to passed *x* co-ordinate.

5.2.3.25 int ppf::datatype::getSysStatus () const [inline]

Returns the system status of the datatype.

Returns:
An integer containing the status value

5.2.3.26 float ppf::datatype::getT (const int *It*) const [inline]

Returns the T vector value at index *It*.

Parameters:
It index in T-direction

Returns:
The T vector value at index *It*

5.2.3.27 string ppf::datatype::getTUnit () const [inline]

Returns the units of the T vector.

Returns:
A string containing the units of the T vector

5.2.3.28 int ppf::datatype::getUserStatus () const [inline]

Returns the user status of the datatype.

Returns:
An integer containing the status value

5.2.3.29 float ppf::datatype::getX (const int *Ix*) const [inline]

Returns the X vector value at index *Ix*.

Parameters:
Ix index in X-direction

Returns:
The X vector value at index *Ix*

5.2.3.30 string ppf::datatype::getXUnit () const [inline]

Returns the units of the X vector.

Returns:
A string containing the units of the X vector

5.2.3.31 void ppf::datatype::readLocal (const string *filename*)

Reads in data from a local file.

Parameters:
filename String containing the filename (including path)

5.2.3.32 void ppf::datatype::resizeT (const int *nt*) [inline]

Resizes the T and data vectors to the specified number of elements along the T axis.

Any contents in the data and T vectors at indicies outside the T range will be lost. If the datatype vectors are initially zero sized and the new T size is > 0 the data vector will be expanded along the X axis by a single element to provide space for 1D data. Setting the T size to zero will destroy the contents of the T, X and data vectors.

Parameters:

nt Number of T elements.

Exceptions:

ppf_error(p. 29) If an invalid number of elements is passed.

5.2.3.33 void ppf::datatype::resizeTX (int *nt*, int *nx*)

Resizes the T, X and data vectors to the specified number of elements along the T and X axes.

Any contents at indicies outside the new T and X ranges will be lost. Setting either the X or T sizes are set to zero the contents of T, X and data vectors will be destroyed.

Parameters:

nt Number of T elements.

nx Number of X elements.

Exceptions:

ppf_error(p. 29) If an invalid number of elements is passed.

5.2.3.34 void ppf::datatype::resizeX (const int *nx*) [inline]

Resizes the X and data vectors to the specified number of elements along the X axis.

Any contents in the data and X vectors at indicies outside the X range will be lost. If the datatype vectors are initially zero sized and the new X size is > 0 the data vector will be expanded along the T axis by a single element to provide space for 1D data. Setting the X size to zero will destroy the contents of the T, X and data vectors.

Parameters:

nx Number of X elements.

Exceptions:

ppf_error(p. 29) If an invalid number of elements is passed.

5.2.3.35 void ppf::datatype::setComment (const string *s*)

Sets the datatype comment The maximum string length is 24 chars, characters beyond this limit will be removed.

Strings shorter than 24 chars will be automatically padded with spaces.

Parameters:
s Comment string.

5.2.3.36 void ppf::datatype::setDataIt (const int *It*, const float *v*) [inline]

Sets the data value at index *It* (*Ix=0*).

If the ppf is 2D then the X index is always taken to be zero.

Parameters:
It index in T-direction
v new data value

5.2.3.37 void ppf::datatype::setDataItIx (const int *It*, const int *Ix*, const float *v*) [inline]

Sets the data value at index [*It*, *Ix*].

Parameters:
It index in T-direction
Ix index in X-direction
v new data value

5.2.3.38 void ppf::datatype::setDataIx (const int *Ix*, const float *v*) [inline]

Sets the data value at index *Ix* (*It=0*).

If the ppf is 2D then the T index is always taken to be zero.

Parameters:
Ix index in X-direction
v new data value

5.2.3.39 void ppf::datatype::setDataUnit (const string *s*)

Sets the units of the data vector The maximum string length is 8 chars, characters beyond this limit will be removed.

Strings shorter than 8 chars will be automatically padded with spaces.

Parameters:
s Data vector unit string.

5.2.3.40 void ppf::datatype::setSysStatus (const int *v*) [inline]

Sets the system status of the datatype.

Parameters:
v Datatype system status flag.

5.2.3.41 void ppf::datatype::setT (const int *It*, const float *v*) [inline]

Sets the T vector value at index *It*.

Parameters:
It index in T-direction

v new T vector value

5.2.3.42 void ppf::datatype::setTUnit (const string *s*)

Sets the units of the T vector.

The maximum string length is 8 chars, characters beyond this limit will be removed. Strings shorter than 8 chars will be automatically padded with spaces.

Parameters:
s T vector unit string.

5.2.3.43 void ppf::datatype::setUserStatus (const int *v*) [inline]

Sets the user status of the datatype.

Parameters:
v Datatype user status flag.

5.2.3.44 void ppf::datatype::setX (const int *Ix*, const float *v*) [inline]

Sets the X vector value at index *Ix*.

Parameters:
Ix index in X-direction

v new X vector value

5.2.3.45 void ppf::datatype::setXUnit (const string *s*)

Sets the units of the X vector The maximum string length is 8 chars, characters beyond this limit will be removed.

Strings shorter than 8 chars will be automatically padded with spaces.

Parameters:
s X vector unit string.

5.2.3.46 int ppf::datatype::sizeT () const [inline]

Returns the T dimension of the data and T vectors.

Returns:
The T dimension of the data and T vectors

5.2.3.47 int ppf::datatype::sizeX () const [inline]

Returns the X dimension of the data and X vectors.

Returns:
The X dimension of the data and X vectors

5.2.3.48 void ppf::datatype::writeLocal (const string *filename*)

Reads in data from a local file.

Parameters:
filename String containing the filename (including path)

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

- Q:/Codes/PPF/FullPPF/ppfdatatype.hpp
- Q:/Codes/PPF/FullPPF/ppfdatatype.cpp

5.3 ppf::ppf_error Class Reference

PPF system exception class.

`#include <ppftypes.hpp>`

Public Member Functions

- **ppf_error** ()
  Basic Constructor.
- **ppf_error** (string errormsg)
  Initialising constructor.
- **ppf_error** (string name, int value)
  Initialising constructor.
- **ppf_error** (string name, int value, string errormsg)
  Initialising constructor.

Public Attributes

- const int **code**
  Holds the PPF system error code.
- const string **method**
  Holds the name of PPF method which failed.
- const string **message**
  Holds a decriptive error message.

5.3.1 Detailed Description

PPF system exception class.

Thrown when an error occurs in the PPF system.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 ppf::ppf_error::ppf_error (string *errormsg*) `[inline]`

Initialising constructor.

Sets error message to passed string. PPF routine and error code are set to null.

Parameters:

errormsg String containing the error message.

5.3.2.2 ppf::ppf_error::ppf_error (string *name*, int *value*) [inline]

Initialising constructor.

Sets PPF routine and error code to passed values. Sets message to a default error message.

Parameters:

name String containing name of PPF routine which failed.

value Integer containing error code returned by failed PPF routine.

5.3.2.3 ppf::ppf_error::ppf_error (string *name*, int *value*, string *errormsg*) [inline]

Initialising constructor.

Sets failed ppf routine name, code and error message to passed values.

Parameters:

name String containing name of PPF routine which failed.

value Integer containing error code returned by failed PPF routine.

errormsg String containing the error message.

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

- Q:/Codes/PPF/FullPPF/ppftypes.hpp

5.4 ppf::system Class Reference

The main PPF system class through which all PPF access is performed.

`#include <ppfsystem.hpp>`

Public Member Functions

- void **connect** () const

  Opens a connection to the public (user = "JETPPF") PPF system for reading only.

- void **connect** (string user) const

  Opens a connection to the PPF system using the supplied user name for reading only.

- void **connect** (string user, const **accessflag** flag) const

  Opens a connection to the PPF system using the supplied parameters.

- void **read** (**datatype** &d, const long shot, string DDA, string dtype) const

  Reads in data for the given shot, DDA and dataype.

- void **read** (**datatype** &d, const long shot, string DDA, string dtype, const **ioflag** flag) const

  Reads in data for the given shot, DDA and dataype using I/O flags.

- void **read** (**datatype** &d, const long shot, const long sequence, string DDA, string dtype) const

  Reads in data for the given shot, sequence, DDA and dataype.

- void **read** (**datatype** &d, const long shot, const long sequence, string DDA, string dtype, const **ioflag** flag) const

  Reads in data for the given shot, sequence, DDA and dataype using I/O flags.

- void **create** (const long shot)

  Creates a new temporary PPF for writing.

- void **create** (const long shot, const long status)

  Creates a new temporary PPF for writing.

- void **create** (const long shot, const string comment)

  Creates a new temporary PPF for writing.

- void **create** (const long shot, const long status, const string comment)

  Creates a new temporary PPF for writing.

- void **write** (**datatype** &d, string DDA, string dtype)

  Writes the provided datatype to an open PPF.

- void **write** (**datatype** &d, string DDA, string dtype, const **ioflag** flag)

  Writes the provided datatype to an open PPF using I/O flags.

- void **closeDDA** () const

  Closes DDA to futher writing.

- void **closeDDA** (const int status) const

  Closes DDA to futher writing.

- void **closeDDA** (string comment) const

  Closes DDA to futher writing.

- void **closeDDA** (const int status, string comment) const

  Closes DDA to futher writing.

- void **abort** () const

  Deletes the temporary PPF without writing it to the PPF system.

- int **commit** () const

  Commits a temporary PPF to the PPF system.

- int **commit** (const string program, const int version) const

  Commits a temporary PPF to the PPF system.

5.4.1 Detailed Description

The main PPF system class through which all PPF access is performed.

5.4.2 Member Function Documentation

5.4.2.1 void ppf::system::abort () const

Deletes the temporary PPF without writing it to the PPF system.

All data written to the temporary PPF will be lost.

Exceptions:

ppf_error(p. 29) thrown if there is a problem, error details are returned in **ppf_error**(p. 29)

5.4.2.2 void ppf::system::closeDDA (const int *status*, string *comment*) const

Closes DDA to futher writing.

A comment and DDA status flag can be provided which will be written to the DDA.

Parameters:

status DDA status flag

comment String holding a comment

Exceptions:

ppf_error(p. 29) thrown if the DDA can not be closed properly, error details are returned in **ppf_error**(p. 29)

5.4.2.3 void ppf::system::closeDDA (string *comment*) const

Closes DDA to futher writing.

A comment can be provided which will be written to the DDA.

Parameters:

comment String holding a comment

Exceptions:

ppf_error(p. 29) thrown if the DDA can not be closed properly, error details are returned in **ppf_error**(p. 29)

5.4.2.4 void ppf::system::closeDDA (const int *status*) const

Closes DDA to futher writing.

A DDA status flag can be provided which will be written to the DDA.

Parameters:

status DDA status flag

Exceptions:

ppf_error(p. 29) thrown if the DDA can not be closed properly, error details are returned in **ppf_error**(p. 29)

5.4.2.5 void ppf::system::closeDDA () const

Closes DDA to futher writing.

Exceptions:

ppf_error(p. 29) thrown if the DDA can not be closed properly, error details are returned in **ppf_error**(p. 29)

5.4.2.6 int ppf::system::commit (const string *program*, const int *version*) const

Commits a temporary PPF to the PPF system.

The temporary PPF will be finalised and written to the PPF system. A string identifying the program used to create the PPF and a version number may be supplied.

Parameters:

program String containing the program name (max. 8 characters)

version Program version number

Exceptions:

ppf_error(p. 29) thrown if the temporary PPF can not be written, error details are returned in **ppf_error**(p. 29)

5.4.2.7 int ppf::system::commit () const

Commits a temporary PPF to the PPF system.

The temporary PPF will be finalised and written to the PPF system.

Exceptions:

ppf_error(p. 29) thrown if the temporary PPF can not be written, error details are returned in **ppf_error**(p. 29)

5.4.2.8 void ppf::system::connect (string *user*, const accessflag *flag*) const

Opens a connection to the PPF system using the supplied parameters.

The access flags are used to determine if the connection is for read access, write access or both. Valid access flags are: *read*, *write*

Parameters:

user String containing the user name to connect with.

flag List of access flags.

5.4.2.9 void ppf::system::connect (string *user*) const

Opens a connection to the PPF system using the supplied user name for reading only.

To open a write connection the full version this routine, connect(const string user, const accessflag flag), must be used.

Parameters:

user A string containing the user name to connect with.

5.4.2.10 void ppf::system::connect () const

Opens a connection to the public (user = "JETPPF") PPF system for reading only.

To use an alternate user name or open a write connection one of the alternate versions of this routine, connect(const string user) or connect(const string user, const accessflag flag), must be used.

5.4.2.11 void ppf::system::create (const long *shot*, const long *status*, const string *comment*)

Creates a new temporary PPF for writing.

Write access is required to create a new PPF. A descriptive text comment and status flag for the PPF can be provided. The new PPF will not be added to the system until at least one datatype is written to it and **commit**()(p. 34) has been called.

Parameters:

shot Shot Number

status PPF system status flag

comment String holding a comment

Exceptions:

ppf_error(p. 29) thrown if an error occurs, error details are returned in **ppf_error**(p. 29)

5.4.2.12 void ppf::system::create (const long *shot*, const string *comment*)

Creates a new temporary PPF for writing.

Write access is required to create a new PPF. A descriptive text comment for the PPF can be provided. The new PPF will not be added to the system until at least one datatype is written to it and **commit**()(p. 34) has been called.

Parameters:

shot Shot Number

comment String holding a comment

Exceptions:

ppf_error(p. 29) thrown if an error occurs, error details are returned in **ppf_error**(p. 29)

5.4.2.13 void ppf::system::create (const long *shot*, const long *status*)

Creates a new temporary PPF for writing.

Write access is required to create a new PPF. A status flag for the PPF can be provided. The new PPF will not be added to the system until at least one datatype is written to it and **commit**()(p. 34) has been called.

Parameters:

shot Shot Number

status PPF system status flag

Exceptions:

ppf_error(p. 29) thrown if an error occurs, error details are returned in **ppf_error**(p. 29)

5.4.2.14 void ppf::system::create (const long *shot*)

Creates a new temporary PPF for writing.

Write access is required to create a new PPF. The new PPF will not be added to the system until at least one datatype is written to it and **commit()**(p. 34) has been called.

Parameters:

shot Shot Number

Exceptions:

ppf_error(p. 29) thrown if an error occurs, error details are returned in **ppf_error**(p. 29)

5.4.2.15 void ppf::system::read (datatype & *d*, const long *shot*, const long *sequence*, string *DDA*, string *dtype*, const ioflag *flag*) const

Reads in data for the given shot, sequence, DDA and datatype using I/O flags.

I/O flags modify the loading of the datatype.

Parameters:

d Datatype object to write data to

shot Shot number

sequence Sequence number

DDA String containing the (up to) 4 character DDA name

dtype String containing the (up to) 4 character datatype name

flag List of I/O flags

Exceptions:

ppf_error(p. 29) thrown if a read error occurs, error details are returned in **ppf_error**(p. 29)

5.4.2.16 void ppf::system::read (datatype & *d*, const long *shot*, const long *sequence*, string *DDA*, string *dtype*) const

Reads in data for the given shot, sequence, DDA and dataype.

Parameters:

d Datatype object to write data to

shot Shot number

sequence Sequence number

DDA String containing the (up to) 4 character DDA name

dtype String containing the (up to) 4 character datatype name

Exceptions:

ppf_error(p. 29) thrown if a read error occurs, error details are returned in **ppf_error**(p. 29)

5.4.2.17 void ppf::system::read (datatype & *d*, const long *shot*, string *DDA*, string *dtype*, const ioflag *flag*) const

Reads in data for the given shot, DDA and dataype using I/O flags.

I/O flags modify the loading of the datatype.

Parameters:

d Datatype object to write data to

shot Shot number

DDA String containing the (up to) 4 character DDA name

dtype String containing the (up to) 4 character datatype name

flag List of I/O flags

Exceptions:

ppf_error(p. 29) thrown if a read error occurs, error details are returned in **ppf_error**(p. 29)

5.4.2.18 void ppf::system::read (datatype & *d*, const long *shot*, string *DDA*, string *dtype*) const

Reads in data for the given shot, DDA and dataype.

Parameters:

d Datatype object to write data to

shot Shot number

DDA String containing the (up to) 4 character DDA name

dtype String containing the (up to) 4 character datatype name

Exceptions:

ppf_error(p. 29) thrown if a read error occurs, error details are returned in **ppf_error**(p. 29)

5.4.2.19 void ppf::system::write (datatype & *d*, string *DDA*, string *dtype*, const ioflag *flag*)

Writes the provided datatype to an open PPF using I/O flags.

Writing can only be performed provided a new temporary PPF has been previously opened via a call to **create(long shot)**(p. 36).

Parameters:

d Datatype object to write to PPF

DDA String containing the (up to) 4 character DDA name

dtype String containing the (up to) 4 character datatype name

flag List of I/O flags

Exceptions:

ppf_error(p. 29) thrown if a write error occurs, error details are returned in **ppf_error**(p. 29)

5.4.2.20 void ppf::system::write (datatype & *d*, string *DDA*, string *dtype*)

Writes the provided datatype to an open PPF.

Writing can only be performed provided a new temporary PPF has been previously opened via a call to **create(long shot)**(p. 36).

Parameters:

d Datatype object to write to PPF

DDA String containing the (up to) 4 character DDA name

dtype String containing the (up to) 4 character datatype name

Exceptions:

ppf_error(p. 29) thrown if a write error occurs, error details are returned in **ppf_error**(p. 29)

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

- Q:/Codes/PPF/FullPPF/ppfsystem.hpp
- Q:/Codes/PPF/FullPPF/ppfsystem.cpp

Index