Fusion-IO Reference

Nathaniel Ferraro

October 5, 2020

Contents

Introduction

The Fusion-IO (FIO) library is an interface for extracting data from the output of various fluid plasma codes in a code-independent way. The library is built to read data directly from the native output of these codes, and therefore no common data format is defined or required. This strategy avoids the loss of information incurred by converting the native code data into a common format (*e.g.* interpolation error), and preserves the properties of the computed solution (*e.g.* divergence-free fields) exactly.

FIO provides access to data through several structures:

Source : provides access to a data file;

Field : allows evaluation of a physical field in real space;

Series : allows evaluation of a time series

Parameter : allows evaluation of specific properties of a field, source, or series.

A *source* structure should always be created first to provide access to a data source, then *fields*, *series*, and *parameters* can be read from that data source and evaluated.

All input and output data representing physical quantities is in SI units.

Field Functions

FIO_CLOSE_FIELD

int ierr = fio_close_field(const int ifield);

call fio_close_field_f(ifield, ierr)
integer, intent(in) :: ifield
integer, intent(out) :: ierr

Parameters

`ifield`: Field handle returned by `fio_get_field`.

Description

Deallocates data associated with field `ifield`

FIO_EVAL_FIELD

int ierr = fio_eval_field(const int ifield, const double* x, double* f);

call fio_eval_field_f(ifield, x, f, ierr) integer, intent(in) :: ifield
real, intent(in), dimension(*) :: x
real, intent(out), dimension(*) :: f
integer, intent(out) :: ierr

Parameters

`ifield`: Field handle returned by `fio_get_field`.
`x`: Three-element array containing (R, φ, Z) coordinates at which to evaluate field.
`f`: Array containing field data. For scalar fields, this is an array of length 1; for vector fields, this is an array of length 3 containing the (R, φ, Z) components of the vector.

Description

Evaluates a field at a given (R, φ, Z) coordinate.

FIO_EVAL_FIELD_DERIV

int ierr = fio_eval_field_deriv(const int ifield, const double* x, double* df);

call fio_eval_field_deriv_f(ifield, x, df, ierr)
integer, intent(in) :: ifield
real, intent(in), dimension(*) :: x
real, intent(out), dimension(*) :: df
integer, intent(out) :: ierr

Parameters

`ifield`: Field handle returned by `fio_get_field`.
`x`: Three-element array containing (R, φ, Z) coordinates at which to evaluate field derivatives.
`df`: Array containing field derivative data data. For scalar fields, this is an array of length 3 containing $(\partial_R f, \partial_\varphi f, \partial_Z f)$; for vector fields, this is an array of length 9 containing $(\partial_R f_R, \partial_R f_\varphi, \partial_R f_Z, \partial_\varphi f_R, \partial_\varphi f_\varphi, \partial_\varphi f_Z, \partial_Z f_R, \partial_Z f_\varphi, \partial_Z f_Z)$.

Description

Evaluates the partial derivatives of a field at a given (R, φ, Z) coordinate. Note that this does NOT return ∇f.

FIO_GET_FIELD

int ierr = fio_get_field(const int isource, const int ifield_type, int* ifield);

call fio_get_field_f(isource, ifield_type, ifield, ierr)
integer, intent(in) :: isource
integer, intent(in) :: ifield_type
integer, intent(out) :: ifield
integer, intent(out) :: ierr

Parameters

`ifield_type`: Field to open. Possibilities include:

Scalar Fields

FIO_SCALAR_POTENTIAL : Scalar potential

FIO_TOTAL_PRESSURE : Total pressure

Species-depdentent Scalar Fields

FIO_PRESSURE : Partial pressure of species

FIO_DENSITY : Number density of species

Vector Fields

FIO_ELECTRIC_FIELD : Electric field

FIO_VECTOR_POTENTIAL : Electromagnetic vector potential

FIO_MAGNETIC_FIELD : Magnetic field

FIO_CURRENT_DENSITY : Current density

FIO_FLUID_VELOCITY : Barycentric velocity

Species-dependent Vector Fields

FIO_VELOCITY : Fluid velocity of species

`ifield`: Field handle returned by `fio_get_field`.

Description

Provides a handle to evaluate field data.

Series Functions

FIO_CLOSE_SERIES

int ierr = fio_close_series(const int iseries);

call fio_close_series_f(iseries, ierr)
integer, intent(in) :: iseries
integer, intent(out) :: ierr

Parameters

`iseries`: Field handle returned by `fio_get_series`.

Description

Deallocates data associated with `iseries`.

FIO_EVAL_SERIES

int ierr = fio_eval_series(const int iseries, const double t, double* s);

call fio_eval_series_f(iseries, t, s, ierr)
integer, intent(in) :: iseries
real, intent(in) :: t
real, intent(out) :: s
integer, intent(out) :: ierr

Parameters

`iseries`: Handle to series returned by `fio_get_series`.
`t`: Time at which to evaluate series.
`s`: Value of series at time `t`.

Description

FIO_GET_SERIES

int ierr = fio_get_series(const int isource, const int iseries_type, int* iseries);

call fio_get_series_f(isource, iseries_type, iseries, ierr)
integer, intent(in) :: isource
integer, intent(in) :: iseries_type
integer, intent(out) :: iseries
integer, intent(out) :: ierr

Parameters

`isource`: The handle to the source file returned by `fio_open_source`.
`iseries_type`: The series to open. Choices include:

LCFS_PSI : Poloidal flux at the last closed flux surface.

MAGAXIS_PSI : Poloidal flux at the magnetic axis.

MAGAXIS_R : R-coordinate of the magnetic axis.

MAGAXIS_Z : Z-coordinate of the magnetic axis.

`iseries`: The handle to the series data.

Description

Returns a handle for accessing data for a time series.

FIO_GET_SERIES_BOUNDS

int ierr = fio_get_series_bounds(const int iseries, double* tmin, double* tmax);

call fio_get_series_bounds_f(iseries, tmin, tmax, ierr)
integer, intent(in) :: iseries
real, intent(out) :: tmin
real, intent(out) :: tmax
integer, intent(out) :: ierr

Parameters

`iseries`: Handle to series returned by `fio_get_series`.
`tmin`: Earliest available time in time series
`tmax`: Latest available time in time series

Description

Returns the bounds of the domain of a time series.

Source Functions

FIO_CLOSE_SOURCE

int ierr = fio_close_source(const int isource);

call fio_close_source_f(isource, ierr)
integer, intent(in) :: isource
integer, intent(out) :: ierr

Parameters

`isource`: Source handle returned by `fio_open_source`.

Description

Closes file and deallocates resources associated with open source.

FIO_OPEN_SOURCE

int ierr = fio_open_source(const int isource_type, const char* filename, int* isource);

call fio_open_source_f(isource_type, filename, isource, ierr)
integer, intent(in) :: isource_type
character(len=*), intent(in) :: filename
integer, intent(out) :: ierr

Parameters

`isource_type`: The type of data source to open. Choices include:

FIO_GATO_SOURCE : (not yet implemented)

FIO_GEQDSK_SOURCE :

FIO_M3DC1_SOURCE :

FIO_MARS_SOURCE : (not yet implemented)

FIO_NIMROD_SOURCE : (not yet implemented)

`filename`: The name of the data source file.
`isource`: The handle to the data source.

Description

Opens a data source.

Not Yet Documented

int fio_get_options(const int); int fio_get_available_fields(const int, int*, int**);
int fio_set_int_option(const int, const int); int fio_set_str_option(const int, const char*);
int fio_set_real_option(const int, const double); int fio_get_int_parameter(const int, const
int, int*); int fio_get_real_parameter(const int, const int, double*); int fio_get_real_field_parameter(const
int, const int, double*);

Error Codes

FIO_SUCCESS (0) : Function call returned without errors.

FIO_UNSUPPORTED (10001) : The requested field, series, parameter is not included in the source data or is not yet implemented in the FIO library.

FIO_OUT_OF_BOUNDS (10002) : The requested coordinates are outside of the domain of the source data.

FIO_FILE_ERROR (10003) : The requested source data could not be succesfully read.

FIO_BAD_DIMENSIONS (10004) :

FIO_BAD_SPECIES (10005) : Data for requested species is not present in source data.

FIO_NO_DATA (10006) : Data expected to be found in the source data was not present.