Field Encapsulation Library

The FEL 2.2 Reference Manual

Patrick Moran Chris Henze

MRJ Technology Solutions
NASA Ames Research Center, M/S T27A-2
Moffett Field, CA, 94035, USA
{pmoran,chenze}@nas.nasa.gov

NAS Technical Report NAS-00-007

January 3, 2000

Public Class Interfaces

This document contains an inventory of the public member functions for each of the FEL classes, it is intended to help the user answer the question:

What member functions can I call for an object of a particular class?

For new users of FEL, and even some experience users of FEL, the answer to this question may occasionally be non-obvious. The C++ inheritance hierarchies for meshes and fields are relatively deep, 6 or 7 levels in many cases, thus there would be many places to search in order to answer the question. Furthermore, the classes that the user instantiates are often leaves in the hierarchy tree, while most methods are inherited from classes closer to the root. This document provides "one stop shopping" if one is seeking a member function summary.

There are some other questions that the user may have that this document is *not* intended to answer. For instance:

Where in the class hierarchy is a particular function implemented?

In general the user should not have to concern himself or herself with where a particular member function is implemented (or reimplemented) in the FEL class hierarchies. If one does need to know, the place to go is the FEL source itself. The FEL release contains an HTML file for each FEL header file. The links in the hypertext files make traversing the class hierarchies relatively painless.

Another question that is answered elsewhere is:

How is a particular member function used?

To avoid having this document grow larger than it already is, we have kept the descriptions associated with each member function brief. Long descriptions, for example for the field member function `at_cell`, would get repeated many times, and this document could expand to telephone book size. The User Guide is a better place to look for discussion on how member functions are intended to be used.

Users who like to learn by writing programs may ask:

What is an example usage of a particular member function?

The place to look for an answer to this question is the FEL primer, or perhaps the User Guide. The primer contains over a dozen examples demonstrating basic aspects of the library. The User Guide contains examples and text on how the functions can be used.

Class: `FEL_absolute_value_field<TO, FROM>`

Description:

Derived field returning absolute values of its component field.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_derived_field` →
`FEL_map_then_interpolate_derived_field1` →
`FEL_absolute_value_field`.

Public Member Functions:

```
FEL_absolute_value_field(FEL_pointer<
  FEL_typed_field<FROM> > f, char* nm = "abso-
  lute_value_field");
```

Construct this built-in derived field.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
```

bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f []) **const;**
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = **FEL_ZONE_UNDEFINED) const;**
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, **FEL_vector3f*) const;**
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, **FEL_time*) const;**
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Typedefs:

`FEL_absolute_value_field<float, float>`
`FEL_absolute_value_of_float_field;`

Class: `FEL_axis`

Description:

Public Member Functions:

Class: `FEL_axis_aligned_mesh`

Description:

FEL_axis_aligned_mesh represents structured meshes where the cells are aligned with the coordinate axes.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_single_mesh` →
`FEL_structured_mesh` →
`FEL_axis_aligned_mesh`.

Public Member Functions:

`const char* get_name() const;`
`void set_name(const char*);`
Access the name of this object.

`int get_n_zones() const;`
Return 1 for any subclass of `FEL_single_mesh`.

`FEL_mesh_ptr get_zone(int) const;`
Return the mesh itself the argument is 0, and NULL otherwise.

`bool inside_bounding_box(const FEL_phys_pos& p);`
Return true if p is inside bounding box of mesh.

`int locate_close_vertex_cell(const FEL_phys_pos&, FEL_vertex_cell*) const;`
Locate a vertex close to the given point.

`bool cell_has_collapsed_edge(const FEL_cell&) const;`
`int volume_of_cell(const FEL_cell&, double*) const;`
`int centroid_of_cell(const FEL_cell&, FEL_vector3f*) const;`
`int longest_edge_length_of_cell(const FEL_cell&, float*) const;`
`int closest_vertex_of_cell(const FEL_phys_pos&, const FEL_cell&, FEL_vertex_cell*) const;`
Get geometric properties of a cell.

`bool is_multi_mesh() const;`
`bool is_curvilinear_surface_mesh() const;`
`bool is_mesh() const;`
Provide run-time type determination for a few mesh subclasses.

`bool is_field() const;`
Provide run-time type determination for a few key FEL types

`int jacobian_at_vertex_cell(const FEL_vertex_cell&, FEL_matrix33f*) const;`
Request Jacobian matrix at given vertex.

```
int contravariant_phys_to_comp_vector(const
  FEL_vertex_cell&, const FEL_vector3f&, FEL_vector3f*)
  const;
int contravariant_phys_to_comp_vector(const
  FEL_structured_pos&, const FEL_vector3f&,
  FEL_vector3f*) const;
```

Convert a contravariant vector in physical space to its equivalent in computational space.

```
bool is_structured_mesh() const;
```

Return true.

```
void set(const FEL_set_keyword_enum, int);
```

Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```

Get a mesh parameter.

```
int card(int k) const;
```

Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```

Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```

Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```

Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```

Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```

Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```

Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```

Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
```

```
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```

Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: `FEL_bitvector`

Description:

The FEL_bitvector class represents—you guessed it—a bitvector.

Public Member Functions:

```
FEL_bitvector(int lo, int hi);
FEL_bitvector(const FEL_bitvector&);
```

Construct a bitvector whose bits can be accessed by the indices lo .. hi.

```
~FEL_bitvector();
```

Destruct a bitvector.

```
bool lookup(int);
void put(int);
void del(int);
void toggle(int);
void complement();
```

Twiddle the bitvector bits.

```
int n() const;
```

Return the number of bits represented.

```
FEL_bitvector& operator&=(const FEL_bitvector&);
FEL_bitvector& operator|=(const FEL_bitvector&);
FEL_bitvector& operator-=(const FEL_bitvector&);
FEL_bitvector& operator^=(const FEL_bitvector&);
```

Apply the bitwise operator over all the corresponding bits in the bitvector.

```
int operator==(const FEL_bitvector& rhs);
int operator!=(const FEL_bitvector &rhs);
```

Test for equality of all the bits of rhs.

```
int operator<=(const FEL_bitvector& rhs);
int operator>=(const FEL_bitvector& rhs);
```

Test if the bits of this are a subset (`<=`) or superset (`>=`) of rhs.

```
int card();
```

Return the number of bits currently set.

```
void reset();
```

Twiddle the bitvector bits.

Class: `FEL_cached_field<T>`

Description:

A wrapper field which caches node query results on another field, and satisfies repeated requests from its cache. FEL_cached_field allocates a core field's worth of memory up front for its potential storage needs (compare with FEL_hash_cached_field).

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_cached_field`.

Public Member Functions:

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global "metadata" associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```

Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f []) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_time_steps() const;
Get number of timesteps associated with this field.

FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());
Construct a new core field where each node has been eagerly evaluated.

int get_min_max(T*, T*, const FEL_time& = FEL_time());
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Typedefs:

FEL_cached_field<float>

FEL_cached_float_field;

FEL_cached_field<FEL_plot3d_q>

FEL_cached_plot3d_q_field;

FEL_cached_field<FEL_vector3f>

FEL_cached_vector3f_field;

Class: `FEL_cell`

Description:

The FEL_cell class represents cells in FEL.

Public Member Functions:

```
FEL_cell();
FEL_cell(FEL_cell_type_enum t);
FEL_cell(FEL_cell_type_enum t, int i, int j, int k);
FEL_cell(const FEL_cell& c, FEL_cell_type_enum t);
FEL_cell(const FEL_cell& c, FEL_cell_type_enum t, short
  si);
FEL_cell(FEL_cell_type_enum t, short si, const
  FEL_vector3i& i, int z, const FEL_time& ti);
```

Initialize an `FEL_cell`.

```
FEL_cell_type_enum get_type() const;
void set_type(FEL_cell_type_enum t);
short get_subid() const;
void set_subid(short si);
FEL_vector3i get_ijk() const;
int get_ijk(int i) const;
void set_ijk(const FEL_vector3i& i);
void set_ijk(int i, int j, int k);
int& operator[](int i);
const int& operator[](int i) const;
int get_unstructured_id() const;
void set_unstructured_id(int i);
int get_zone() const;
void set_zone(int z);
```

Access the cell data members.

```
float get_physical_time() const;
float get_computational_time() const;
int get_time_step() const;
FEL_time get_time() const;
void set_time(const FEL_time& t);
void set_physical_time(float t);
void set_computational_time(float t);
void set_time_step(int t);
void set_time_undefined();
```

Access the time representation.

```
friend ostream& operator<<(ostream&, const FEL_cell&);
```

Output a cell representation to an ostream.

```
int get_dim() const;
```

Get the dimension of the cell.

```
int get_n_nodes() const;
```

`int get_n_vertices() const;`

Get the number of nodes, or the number of vertices.

`bool simplicial() const;`

Return true if the cell is a vertex, edge, triangle or tetrahedron.

`friend bool operator==(const FEL_cell& lhs, const FEL_cell& rhs);`
`friend bool operator!=(const FEL_cell& lhs, const FEL_cell& rhs);`

Test two cells for equality by comparing corresponding data members.

Class: `FEL_cell_interpolant`

Description:

An FEL_cell_interpolant contains public cell and interpolant data members.

Public Member Functions:

`friend ostream& operator<<(ostream&, const FEL_cell_interpolant&);`

Write to an ostream.

Class: `FEL_cell_iter`

Description:

A class for iterating over various types of cells.

Public Member Functions:

`bool done() const;`

Test whether the iterator is done.

`friend bool operator!=(const FEL_cell_iter& lhs, const FEL_cell_iter& rhs);`

Compare an iterator with an iterator initialized by end().

`const FEL_cell& operator*();`

Dereference the iterator.

`void operator++();`
`void operator++(int);`

Increment the iterator.

`int get_zone();`
`void set_zone(int zn);`

Access the cell zone.

`void set_physical_time(float pt);`
`void set_computational_time(float ct);`
`FEL_time get_time();`

Access the time of the cell.

`int card() const;`

Return the number of cells that this iterator is initialized to produce (not supported if the iterator is initialized to loop over mulitiple zones).

`void set_time(const FEL_time& t);`
`void set_time_step(int t);`

Access the time of the cell.

Class: `FEL_component_field<TO, FROM>`

Description:

Derived field returning the result of the [] operator on its component field. Typically used to retrieve the i'th component of a vector.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_derived_field` →
`FEL_map_then_interpolate_derived_field1` →
`FEL_component_field`.

Public Member Functions:

```
FEL_component_field(FEL_pointer< FEL_typed_field<FROM> >
  f, int c, char* nm = "FEL_component_field");
```
Construct this built-in derived field.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```
Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```
Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```
Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```
Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
```

bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f [ ]) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Typedefs:

`FEL_component_field<float, FEL_plot3d_q>`
`FEL_component_of_plot3d_q_field;`

`FEL_component_field<float, FEL_vector3f>`
`FEL_component_of_vector3f_field;`

Class: `FEL_constant_field<T>`

Description:

A field which returns the same (constant) value from all locations. Useful for scaling operations, or other algebraic manipulations on fields, using the derived field facility.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_constant_field`.

Public Member Functions:

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```

Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f []) **const;**
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_time_steps() const;
Get number of timesteps associated with this field.

FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());
Construct a new core field where each node has been eagerly evaluated.

int get_min_max(T*, T*, const FEL_time& = FEL_time());
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Typedefs:

FEL_constant_field<float>

FEL_constant_float_field;

FEL_constant_field<FEL_matrix33f>

FEL_constant_matrix33f_field;

FEL_constant_field<FEL_vector3f>

FEL_constant_vector3f_field;

Class: FEL_core_field<T>

Description:

A field whose node values reside in memory (in contrast to a derived field, for which values are generated only on demand).

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_field →
FEL_typed_field →
FEL_core_field.

Public Member Functions:

```
FEL_core_field(FEL_mesh_ptr, T*, char* = "core_field");
FEL_core_field(FEL_mesh_ptr, T*, bool, char* =
  "core_field");
FEL_core_field(FEL_mesh_ptr, FEL_pointer<
  FEL_core_field<T> >, char* = "shared_core_field");
```

Three options for constructing a core field.

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f []) **const;**
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_time_steps() const;
Get number of timesteps associated with this field.

FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());
Construct a new core field where each node has been eagerly evaluated.

int get_min_max(T*, T*, const FEL_time& = FEL_time());

Get the minimum and maximum values of the field (undefined for non-scalar fields).

Typedefs:

`FEL_core_field<float>`
`FEL_core_float_field;`

`FEL_core_field<FEL_plot3d_q>`
`FEL_core_plot3d_q_field;`

`FEL_core_field<FEL_vector3f>`
`FEL_core_vector3f_field;`

Class: FEL_cross_field<TO, FROM1, FROM2>

Description:

Derived field returning the cross product of its component fields.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_field →
FEL_typed_field →
FEL_derived_field →
FEL_map_then_interpolate_derived_field2 →
FEL_cross_field.

Public Member Functions:

```
FEL_cross_field(FEL_pointer< FEL_typed_field<FROM1> >
  f1, FEL_pointer< FEL_typed_field<FROM2> > f2, char* nm
  = "cross_field");
```
Construct this built-in derived field.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```
Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```
Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```
Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```
Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
```

bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f [ ]) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Typedefs:

`FEL_cross_field<FEL_vector3f, FEL_vector3f, FEL_vector3f>`
`FEL_cross_of_vector3f_field;`

Class: `FEL_curl_field1<TO, FROM>`

Description:

Class supporting first-order accurate estimation of curl.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_differential_operator_field` →
`FEL_differential_operator_field1` →
`FEL_curl_field1`.

Public Member Functions:

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

`FEL_mesh_ptr get_mesh() const;`
Retrieve a pointer to the mesh associated with this field.

`int card(int d) const;`
Get the number of cells of specified dimension d.

`int coordinates_at_cell(const FEL_cell&, FEL_vector3f [])`
`const;`
Get physical-space coordinates of specified cell.

`void begin(FEL_vertex_cell_iter* vi) const;`
Initialize a `vertex_cell` iterator to loop over all vertices.

`void begin(FEL_cell_iter* ci) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_vertex_cell_iter* vi, int k, ...) const;`
`int begin(FEL_cell_iter* ci, int k, ...) const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter* ci) const;`
Initialize an iterator to test if at end.

`bool is_mesh() const;`
Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Typedefs:

FEL_curl_field1<FEL_vector3f, FEL_vector3f>
FEL_curl_of_vector3f_field1;

Class: `FEL_curl_field2<TO, FROM>`

Description:

Class supporting second-order accurate estimation of curl..

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_differential_operator_field` →
`FEL_differential_operator_field2` →
`FEL_curl_field2`.

Public Member Functions:

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

`FEL_mesh_ptr get_mesh() const;`
Retrieve a pointer to the mesh associated with this field.

`int card(int d) const;`
Get the number of cells of specified dimension d.

`int coordinates_at_cell(const FEL_cell&, FEL_vector3f [])  const;`
Get physical-space coordinates of specified cell.

`void begin(FEL_vertex_cell_iter* vi) const;`
Initialize a `vertex_cell` iterator to loop over all vertices.

`void begin(FEL_cell_iter* ci) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_vertex_cell_iter* vi, int k, ...)  const;`
`int begin(FEL_cell_iter* ci, int k, ...)  const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter* ci) const;`
Initialize an iterator to test if at end.

`bool is_mesh() const;`
Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Typedefs:

FEL_curl_field2<FEL_vector3f, FEL_vector3f>
FEL_curl_of_vector3f_field2;

Class: `FEL_curl_interpolator<TO, FROM>`

Description:

This class calulates the curl by first-order accurate methods, getting the necessary derivatives by analytic differentiation of either isoparametric or physical space interpolation polynomials.

Inheritance Hierarchy:

`FEL_derivative_interpolator` $\rightarrow$
`FEL_curl_interpolator`.

Public Member Functions:

`int interpolate(const FEL_interpolant*, const FROM&, const FROM&, const FROM&, const FROM&, TOO[])=0;`
Interface for derivative interpolation on a tetrahedron.

`int interpolate(const FEL_interpolant*, const FEL_mesh*, const FEL_cell&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, TOO[])=0;`
Interface for derivative interpolation on a hexahedron.

`int interpolate(const FEL_vector3f&, const FEL_interpolant*, const FROM&, const FROM&, const FROM&, const FROM&, TO*);`
Calculate the curl at an interior point (physical coords, in `FEL_vector3f`) of a tetrahedron, whose node values are specified by the `FROM`'s, according to interpolation scheme specified by the `FEL_interpolant`, and put answer in `TO*`.

`int interpolate(const FEL_vector3f&, const FEL_interpolant*, const int&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, TO*);`
Calculate the curl at an interior point (physical coords, in `FEL_vector3f`) of a hexahedron, whose node values are specified by the `FROM`'s, according to interpolation scheme specified by the `FEL_interpolant`, and put answer in `TO*`.

`int interpolate(const FEL_interpolant*, const FROM&, const FROM&, const FROM&, const FROM&, TO[]);`
Calculate the curl at all vertices of a tetrahedron, according to interpolation scheme specified by `FEL_interpolant`, and put answers in `TO[]`.

`int interpolate(const FEL_interpolant*, const FEL_mesh*, const FEL_cell&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, TO[]);`
Calculate the curl at all vertices of a hexahedron, according to interpolation scheme specified by `FEL_interpolant`, and put answers in `TO[]`.

Class: `FEL_curvilinear_mesh`

Description:

FEL_curvilinear_mesh is an abstract class that is the parent to all the curvilinear mesh subclasses.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_single_mesh` →
`FEL_structured_mesh` →
`FEL_curvilinear_mesh`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
int get_n_zones() const;
```
Return 1 for any subclass of `FEL_single_mesh`.

```
FEL_mesh_ptr get_zone(int) const;
```
Return the mesh itself the argument is 0, and NULL otherwise.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
int jacobian_at_vertex_cell(const FEL_vertex_cell&,
  FEL_matrix33f*) const;
```
Request Jacobian matrix at given vertex.

```
int contravariant_phys_to_comp_vector(const
  FEL_vertex_cell&, const FEL_vector3f&, FEL_vector3f*)
  const;
int contravariant_phys_to_comp_vector(const
  FEL_structured_pos&, const FEL_vector3f&,
  FEL_vector3f*) const;
```

Convert a contravariant vector in physical space to its equivalent in computational space.

```
bool is_structured_mesh() const;
```

Return true.

```
void set(const FEL_set_keyword_enum, int);
```

Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```

Get a mesh parameter.

```
int card(int k) const;
```

Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```

Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```

Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```

Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```

Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```

Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```

Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```

Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
```

```
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```
Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```
Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```
Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```
Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```
Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```
Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

Class: `FEL_curvilinear_mesh_time_series_layout`

Description:

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_single_mesh` →
`FEL_structured_mesh` →
`FEL_curvilinear_mesh` →
`FEL_curvilinear_mesh_time_series_layout`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
int get_n_zones() const;
```
Return 1 for any subclass of `FEL_single_mesh`.

```
FEL_mesh_ptr get_zone(int) const;
```
Return the mesh itself the argument is 0, and NULL otherwise.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
int jacobian_at_vertex_cell(const FEL_vertex_cell&,
  FEL_matrix33f*) const;
```
Request Jacobian matrix at given vertex.

```
int contravariant_phys_to_comp_vector(const
  FEL_vertex_cell&, const FEL_vector3f&, FEL_vector3f*)
  const;
int contravariant_phys_to_comp_vector(const
  FEL_structured_pos&, const FEL_vector3f&,
  FEL_vector3f*) const;
```

Convert a contravariant vector in physical space to its equivalent in computational space.

```
bool is_structured_mesh() const;
```

Return true.

```
void set(const FEL_set_keyword_enum, int);
```

Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```

Get a mesh parameter.

```
int card(int k) const;
```

Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```

Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```

Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```

Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```

Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```

Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```

Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```

Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
```

```
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```

Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: `FEL_curvilinear_mesh_xyz_layout`

Description:

FEL_curvilinear_mesh_xyz_layout represents curvilinear meshes where the coordinates are stored in an FEL_vector3f array, and IBLANK values are all assumed to be 1.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_single_mesh` →
`FEL_structured_mesh` →
`FEL_curvilinear_mesh` →
`FEL_curvilinear_mesh_xyz_layout`.

Public Member Functions:

```
FEL_curvilinear_mesh_xyz_layout(int d0, int d1, int
  d2, FEL_vector3f* xyz, const char* nm = "curvilin-
  ear_mesh_xyz_layout");
```
Construct a curvilinear mesh with dimensions d0, d1 and d2. Use the xyz buffer for coordinates data.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
int get_n_zones() const;
```
Return 1 for any subclass of `FEL_single_mesh`.

```
FEL_mesh_ptr get_zone(int) const;
```
Return the mesh itself the argument is 0, and NULL otherwise.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
int jacobian_at_vertex_cell(const FEL_vertex_cell&,
  FEL_matrix33f*) const;
```
Request Jacobian matrix at given vertex.

```
int contravariant_phys_to_comp_vector(const
  FEL_vertex_cell&, const FEL_vector3f&, FEL_vector3f*)
  const;
int contravariant_phys_to_comp_vector(const
  FEL_structured_pos&, const FEL_vector3f&,
  FEL_vector3f*) const;
```
Convert a contravariant vector in physical space to its equivalent in computational space.

```
bool is_structured_mesh() const;
```
Return true.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```
Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```
Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```
Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```
Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```
Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```
Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

Class: `FEL_curvilinear_mesh_xyzi_field_layout`

Description:

FEL_curvilinear_mesh_xyzi_field_layout represents curvilinear meshes where the coordinates and IBLANK values come from the field provided at construction time.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_single_mesh` →
`FEL_structured_mesh` →
`FEL_curvilinear_mesh` →
`FEL_curvilinear_mesh_xyzi_field_layout`.

Public Member Functions:

```
FEL_curvilinear_mesh_xyzi_field_layout(int d0, int d1,
  int d2, FEL_vector3f_and_int_field_ptr f, const char*
  nm = "curvilinear_mesh_xyzi_field_layout");
```
Construct a curvilinear mesh with dimensions d0, d1 and d2. Consult field f for coordinate and IBLANK data.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
int get_n_zones() const;
```
Return 1 for any subclass of `FEL_single_mesh`.

```
FEL_mesh_ptr get_zone(int) const;
```
Return the mesh itself the argument is 0, and NULL otherwise.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
int jacobian_at_vertex_cell(const FEL_vertex_cell&,
  FEL_matrix33f*) const;
```
Request Jacobian matrix at given vertex.

```
int contravariant_phys_to_comp_vector(const
  FEL_vertex_cell&, const FEL_vector3f&, FEL_vector3f*)
  const;
int contravariant_phys_to_comp_vector(const
  FEL_structured_pos&, const FEL_vector3f&,
  FEL_vector3f*) const;
```
Convert a contravariant vector in physical space to its equivalent in computational space.

```
bool is_structured_mesh() const;
```
Return true.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
```

```
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```
Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```
Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```
Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```
Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```
Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```
Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

Class: `FEL_curvilinear_mesh_xyzi_layout`

Description:

FEL_curvilinear_mesh_xyzi_layout represents curvilinear meshes where the coordinates and IBLANK values are stored in an FEL_vector3f_and_int array.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_single_mesh` →
`FEL_structured_mesh` →
`FEL_curvilinear_mesh` →
`FEL_curvilinear_mesh_xyzi_layout`.

Public Member Functions:

`FEL_curvilinear_mesh_xyzi_layout(int d0, int d1, int d2, FEL_vector3f_and_int* xyzi, const char* nm = "curvilinear_mesh_xyzi_layout");`

Construct a curvilinear mesh with dimensions d0, d1 and d2. Use the xyzi buffer for coordinates and IBLANK data.

`const char* get_name() const;`
`void set_name(const char*);`

Access the name of this object.

`int get_n_zones() const;`

Return 1 for any subclass of `FEL_single_mesh`.

`FEL_mesh_ptr get_zone(int) const;`

Return the mesh itself the argument is 0, and NULL otherwise.

`bool inside_bounding_box(const FEL_phys_pos& p);`

Return true if p is inside bounding box of mesh.

`int locate_close_vertex_cell(const FEL_phys_pos&, FEL_vertex_cell*) const;`

Locate a vertex close to the given point.

`bool cell_has_collapsed_edge(const FEL_cell&) const;`
`int volume_of_cell(const FEL_cell&, double*) const;`
`int centroid_of_cell(const FEL_cell&, FEL_vector3f*) const;`
`int longest_edge_length_of_cell(const FEL_cell&, float*) const;`
`int closest_vertex_of_cell(const FEL_phys_pos&, const FEL_cell&, FEL_vertex_cell*) const;`

Get geometric properties of a cell.

`bool is_multi_mesh() const;`
`bool is_curvilinear_surface_mesh() const;`
`bool is_mesh() const;`

Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
int jacobian_at_vertex_cell(const FEL_vertex_cell&,
  FEL_matrix33f*) const;
```
Request Jacobian matrix at given vertex.

```
int contravariant_phys_to_comp_vector(const
  FEL_vertex_cell&, const FEL_vector3f&, FEL_vector3f*)
  const;
int contravariant_phys_to_comp_vector(const
  FEL_structured_pos&, const FEL_vector3f&,
  FEL_vector3f*) const;
```
Convert a contravariant vector in physical space to its equivalent in computational space.

```
bool is_structured_mesh() const;
```
Return true.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
```

```
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```
Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```
Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```
Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```
Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```
Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```
Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

Class: FEL_curvilinear_surface_mesh

Description:

FEL_curvilinear_surface_mesh class is used to represent surfaces in 3-space.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_mesh →
FEL_single_mesh →
FEL_structured_mesh →
FEL_curvilinear_mesh →
FEL_curvilinear_surface_mesh.

Public Member Functions:

```
FEL_curvilinear_surface_mesh(int, int, FEL_vector3f*,
  const char* = name_default);
FEL_curvilinear_surface_mesh(int, int, int,
  FEL_vector3f*, const char* = name_default);
FEL_curvilinear_surface_mesh(int, int,
  FEL_vector3f_and_int*, const char* = name_default);
FEL_curvilinear_surface_mesh(int, int, int,
  FEL_vector3f_and_int*, const char* = name_default);
```

Construct a curvilinear surface mesh.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
int get_n_zones() const;
```

Return 1 for any subclass of FEL_single_mesh.

```
FEL_mesh_ptr get_zone(int) const;
```

Return the mesh itself the argument is 0, and NULL otherwise.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```

Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```

Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```

Get geometric properties of a cell.

```
bool is_multi_mesh() const;
```

```
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
int jacobian_at_vertex_cell(const FEL_vertex_cell&,
  FEL_matrix33f*) const;
```
Request Jacobian matrix at given vertex.

```
int contravariant_phys_to_comp_vector(const
  FEL_vertex_cell&, const FEL_vector3f&, FEL_vector3f*)
  const;
int contravariant_phys_to_comp_vector(const
  FEL_structured_pos&, const FEL_vector3f&,
  FEL_vector3f*) const;
```
Convert a contravariant vector in physical space to its equivalent in computational space.

```
bool is_structured_mesh() const;
```
Return true.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```

Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```

Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```

Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: `FEL_derivative_interpolator<TO, FROM>`

Description:

Abstract base class for the ”derivative interpolators”, which actually apply the various first-order differential operators.

Public Member Functions:

```
int interpolate(const FEL_interpolant*, const FROM&,
  const FROM&, const FROM&, const FROM&, TO[])=0;
```

Interface for derivative interpolation on a tetrahedron.

```
int interpolate(const FEL_interpolant*, const
  FEL_mesh*, const FEL_cell&, const FROM&, const FROM&,
  const FROM&, const FROM&, const FROM&, const FROM&,
  const FROM&, const FROM&, TO[])=0;
```

Interface for derivative interpolation on a hexahedron.

Class: `FEL_derived_field<TO>`

Description:

Pure virtual base class for derived fields

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_derived_field`.

Public Member Functions:

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```

Get the number of cells of specified dimension d.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f
  []) const;
```

Get physical-space coordinates of specified cell.

`void begin(FEL_vertex_cell_iter* vi) const;`
Initialize a `vertex_cell` iterator to loop over all vertices.

`void begin(FEL_cell_iter* ci) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_vertex_cell_iter* vi, int k, ...) const;`
`int begin(FEL_cell_iter* ci, int k, ...) const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter* ci) const;`
Initialize an iterator to test if at end.

`bool is_mesh() const;`
Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

`FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());`
Construct a new core field where each node has been eagerly evaluated.

`int get_min_max(T*, T*, const FEL_time& = FEL_time());`
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Class: `FEL_difference_field<TO, FROM1, FROM2>`

Description:

Derived field returning the difference between its component fields.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_derived_field` →
`FEL_map_then_interpolate_derived_field2` →
`FEL_difference_field`.

Public Member Functions:

```
FEL_difference_field(FEL_pointer< FEL_typed_field<FROM1>
  > f1, FEL_pointer< FEL_typed_field<FROM2> > f2, char*
  nm = "difference_field");
```
Construct this built-in derived field.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```
Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```
Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```
Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```
Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
```

bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f [ ]) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_time_steps() const;
Get number of timesteps associated with this field.

Typedefs:

FEL_difference_field<float, float, float>
FEL_difference_of_float_field;

FEL_difference_field<FEL_vector3f, FEL_vector3f, FEL_vector3f>
FEL_difference_of_vector3f_field;

Class: `FEL_differential_operator_field<TO, FROM>`

Description:

Base class for differential operator fields.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_differential_operator_field`.

Public Member Functions:

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f [ ]) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_time_steps() const;
Get number of timesteps associated with this field.

Class: `FEL_differential_operator_field1<TO, FROM>`

Description:

Base class for first-order differential operator fields.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_differential_operator_field` →
`FEL_differential_operator_field1`.

Public Member Functions:

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f [ ]) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_time_steps() const;
Get number of timesteps associated with this field.

Class: `FEL_differential_operator_field2<TO, FROM>`

Description:

Base class for second-order differential operator fields.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_differential_operator_field` →
`FEL_differential_operator_field2`.

Public Member Functions:

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

`FEL_mesh_ptr get_mesh() const;`
Retrieve a pointer to the mesh associated with this field.

`int card(int d) const;`
Get the number of cells of specified dimension d.

`int coordinates_at_cell(const FEL_cell&, FEL_vector3f []) const;`
Get physical-space coordinates of specified cell.

`void begin(FEL_vertex_cell_iter* vi) const;`
Initialize a `vertex_cell` iterator to loop over all vertices.

`void begin(FEL_cell_iter* ci) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_vertex_cell_iter* vi, int k, ...) const;`
`int begin(FEL_cell_iter* ci, int k, ...) const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter* ci) const;`
Initialize an iterator to test if at end.

`bool is_mesh() const;`
Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Class: `FEL_divergence_field1<TO, FROM>`

Description:

Class supporting first-order accurate estimation of divergence.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_differential_operator_field` →
`FEL_differential_operator_field1` →
`FEL_divergence_field1`.

Public Member Functions:

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

`FEL_mesh_ptr get_mesh() const;`
Retrieve a pointer to the mesh associated with this field.

`int card(int d) const;`
Get the number of cells of specified dimension d.

`int coordinates_at_cell(const FEL_cell&, FEL_vector3f [])  const;`
Get physical-space coordinates of specified cell.

`void begin(FEL_vertex_cell_iter* vi) const;`
Initialize a `vertex_cell` iterator to loop over all vertices.

`void begin(FEL_cell_iter* ci) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_vertex_cell_iter* vi, int k, ...)  const;`
`int begin(FEL_cell_iter* ci, int k, ...)  const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter* ci) const;`
Initialize an iterator to test if at end.

`bool is_mesh() const;`
Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Typedefs:

FEL_divergence_field1<double, FEL_vector3d>
FEL_divergence_of_vector3d_field1;

FEL_divergence_field1<float, FEL_vector3f>
FEL_divergence_of_vector3f_field1;

Class: `FEL_divergence_field2<TO, FROM>`

Description:

Class supporting second-order accurate estimation of divergence.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_differential_operator_field` →
`FEL_differential_operator_field2` →
`FEL_divergence_field2`.

Public Member Functions:

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

`FEL_mesh_ptr get_mesh() const;`
Retrieve a pointer to the mesh associated with this field.

`int card(int d) const;`
Get the number of cells of specified dimension d.

`int coordinates_at_cell(const FEL_cell&, FEL_vector3f [])  const;`
Get physical-space coordinates of specified cell.

`void begin(FEL_vertex_cell_iter* vi) const;`
Initialize a `vertex_cell` iterator to loop over all vertices.

`void begin(FEL_cell_iter* ci) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_vertex_cell_iter* vi, int k, ...)  const;`
`int begin(FEL_cell_iter* ci, int k, ...)  const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter* ci) const;`
Initialize an iterator to test if at end.

`bool is_mesh() const;`
Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Typedefs:

FEL_divergence_field2<double, FEL_vector3d>
FEL_divergence_of_vector3d_field2;

FEL_divergence_field2<float, FEL_vector3f>
FEL_divergence_of_vector3f_field2;

Class: `FEL_divergence_interpolator<TO, FROM>`

Description:

This class calulates the divergence by first-order accurate methods, getting the necessary derivatives by analytic differentiation of either isoparametric or physical space interpolation polynomials.

Inheritance Hierarchy:

`FEL_derivative_interpolator` → `FEL_divergence_interpolator`.

Public Member Functions:

`int interpolate(const FEL_interpolant*, const FROM&, const FROM&, const FROM&, const FROM&, TOO[])=0;`

Interface for derivative interpolation on a tetrahedron.

`int interpolate(const FEL_interpolant*, const FEL_mesh*, const FEL_cell&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, TOO[])=0;`

Interface for derivative interpolation on a hexahedron.

`int interpolate(const FEL_vector3f&, const FEL_interpolant*, const FROM&, const FROM&, const FROM&, const FROM&, TO*);`

Calculate the divergence at an interior point (physical coords, in `FEL_vector3f`) of a tetrahedron, whose node values are specified by the `FROM`'s, according to interpolation scheme specified by the `FEL_interpolant`, and put answer in `TO*`.

`int interpolate(const FEL_vector3f&, const FEL_interpolant*, const int&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, TO*);`

Calculate the divergence at an interior point (physical coords, in `FEL_vector3f`) of a hexahedron, whose node values are specified by the `FROM`'s, according to interpolation scheme specified by the `FEL_interpolant`, and put answer in `TO*`.

`int interpolate(const FEL_interpolant*, const FROM&, const FROM&, const FROM&, const FROM&, TO[]);`

Calculate the divergence at all vertices of a tetrahedron, according to interpolation scheme specified by `FEL_interpolant`, and put answers in `TO[]`.

`int interpolate(const FEL_interpolant*, const FEL_mesh*, const FEL_cell&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, TO[]);`

Calculate the divergence at all vertices of a hexahedron, according to interpolation scheme specified by `FEL_interpolant`, and put answers in `TO[]`.

Class: FEL_dot_field<TO, FROM1, FROM2>

Description:

Derived field returning the inner or "dot" product of its component fields.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_field →
FEL_typed_field →
FEL_derived_field →
FEL_map_then_interpolate_derived_field2 →
FEL_dot_field.

Public Member Functions:

```
FEL_dot_field(FEL_pointer< FEL_typed_field<FROM1> > f1,
  FEL_pointer< FEL_typed_field<FROM2> > f2, char* nm =
  "dot_field");
```
Construct this built-in derived field.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```
Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```
Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```
Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```
Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
```

bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f [ ]) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Typedefs:

`FEL_dot_field<float, FEL_vector3f, FEL_vector3f>`
`FEL_dot_of_vector3f_field;`

Class: `FEL_edge_vertices`

Description:

FEL_edge_vertices is used to represent the 2 integer vertex indices of an edge. The class is used by FEL_unstructured_mesh.

Inheritance Hierarchy:

`FEL_vector2i` →
`FEL_edge_vertices`.

Public Member Functions:

`FEL_edge_vertices(int v0, int v1);`

Initialize.

`friend unsigned FEL_hash(const FEL_edge_vertices& ev);`

Return a cheap hash.

Class: `FEL_field`

Description:

Pure virtual base class for all fields. This class provides a common interface to the underlying mesh; and also declares various field ”type information functions” that are defined by its subclasses.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```
Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```
Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```
Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```
Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```
Get the number of cells of specified dimension d.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f
  []) const;
```
Get physical-space coordinates of specified cell.

```
void begin(FEL_vertex_cell_iter* vi) const;
```
Initialize a `vertex_cell` iterator to loop over all vertices.

```
void begin(FEL_cell_iter* ci) const;
```
Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_vertex_cell_iter* vi, int k, ...)  const;
int begin(FEL_cell_iter* ci, int k, ...)  const;
```
Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter* ci) const;
```
Initialize an iterator to test if at end.

```
bool is_mesh() const;
```

Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Class: `FEL_fixed_interval_time_series_field<T>`

Description:

FEL_fixed_interval_time_series_field represents time-varying fields where there is a fixed time interval between each pair of consecutive time steps.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_time_varying_field` →
`FEL_time_series_field` →
`FEL_fixed_interval_time_series_field`.

Public Member Functions:

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```

Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f []) **const;**
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

FEL_pointer< FEL_typed_field<T> > get_field(int t);
Get the field associated with a time step t.

bool load(int);
bool load_all();
void unload(int);
void unload_all();
void unload_least_recently_used();
void set_working_set_size(int);
void set_verbose(bool b);
void show_working_set(ostream&);
Manage the working set directly.

bool is_time_series_field() const;
Return true.

`int get_n_time_steps() const;`
Return the number of time steps in the time series.

`FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());`
Construct a new core field where each node has been eagerly evaluated.

`int get_min_max(T*, T*, const FEL_time& = FEL_time());`
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Typedefs:

`FEL_fixed_interval_time_series_field<float>`
`FEL_fixed_interval_time_series_float_field;`

`FEL_fixed_interval_time_series_field<FEL_plot3d_q>`
`FEL_fixed_interval_time_series_plot3d_q_field;`

`FEL_fixed_interval_time_series_field<FEL_vector3f_and_int>`
`FEL_fixed_interval_time_series_vector3f_and_int_field;`

`FEL_fixed_interval_time_series_field<FEL_vector3f>`
`FEL_fixed_interval_time_series_vector3f_field;`

Class: `FEL_fixed_interval_time_series_plot3d_field`

Description:
Field manager for one type of unsteady field.

Inheritance Hierarchy:
`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_plot3d_field` →
`FEL_fixed_interval_time_series_plot3d_field`.

Public Member Functions:

```
FEL_fixed_interval_time_series_plot3d_field(char*, int,
  FEL_plot3d_filename_callback, void*, float, float,
  unsigned = 0);
FEL_fixed_interval_time_series_plot3d_field(FEL_mesh_ptr,
  int, FEL_plot3d_filename_callback, void*, float,
  float, unsigned = 0);
```
creates a field "manager" for an unsteady field.

```
FEL_mesh_ptr get_mesh();
FEL_plot3d_q_field_ptr get_q_field();
FEL_plot3d_density_momentum_field_ptr
  get_density_momentum_field();
FEL_vector3f_field_ptr get_momentum_field();
FEL_float_field_ptr get_density_field();
FEL_float_field_ptr get_energy_field();
```
Retrieve mesh, or PLOT3D primitive fields.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_float_field_ptr make_float_field(FEL_float_field_enum);
FEL_vector3f_field_ptr make_vector3f_field(FEL_vector3f_field_enum);
```
Request a derived float field or vector field from the manager. The complete list of enums can be found in the FEL User Guide, in `FEL_plot3d_field.h`, or the PLOT3D User's Manual (as PLOT3D "function numbers", which can be used directly.

```
bool is_mesh() const;
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
void set(const FEL_set_keyword_enum, int);
```
Set an option on mesh or any field associated with the manager.

Class: `FEL_globus_remote_field<T>`

Description:

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_globus_remote_field`.

Public Member Functions:

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```

Get the number of cells of specified dimension d.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f
  []) const;
```

Get physical-space coordinates of specified cell.

`void begin(FEL_vertex_cell_iter* vi) const;`
Initialize a `vertex_cell` iterator to loop over all vertices.

`void begin(FEL_cell_iter* ci) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_vertex_cell_iter* vi, int k, ...) const;`
`int begin(FEL_cell_iter* ci, int k, ...) const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter* ci) const;`
Initialize an iterator to test if at end.

`bool is_mesh() const;`
Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

`FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());`
Construct a new core field where each node has been eagerly evaluated.

`int get_min_max(T*, T*, const FEL_time& = FEL_time());`
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Typedefs:

`FEL_globus_remote_field<float>`
`FEL_globus_remote_float_field;`

FEL_globus_remote_field<FEL_plot3d_q>
FEL_globus_remote_plot3d_q_field;

FEL_globus_remote_field<FEL_vector3f>
FEL_globus_remote_vector3f_field;

Class: `FEL_globus_remote_mesh`

Description:

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_globus_remote_mesh`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
bool is_structured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
```

`int coordinates_and_iblank_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f_and_int*) const;`
Access coordinates and IBLANK.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Convert time representation.

`void begin(FEL_vertex_cell_iter*) const;`
Initialize a `vertex_cell` iterator to loop over all the vertices.

`void begin(FEL_cell_iter*) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_cell_iter*, int, ...) const;`
`int begin(FEL_vertex_cell_iter*, int, ...) const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter*) const;`
Initialize an iterator to test if at end.

`bool is_unstructured_mesh() const;`
Provide run-time type determination for a few mesh subclasses.

Class: `FEL_gradient_field1<TO, FROM>`

Description:

Class supporting first-order accurate estimation of gradients.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_differential_operator_field` →
`FEL_differential_operator_field1` →
`FEL_gradient_field1`.

Public Member Functions:

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

`FEL_mesh_ptr get_mesh() const;`
Retrieve a pointer to the mesh associated with this field.

`int card(int d) const;`
Get the number of cells of specified dimension d.

`int coordinates_at_cell(const FEL_cell&, FEL_vector3f [])  const;`
Get physical-space coordinates of specified cell.

`void begin(FEL_vertex_cell_iter* vi) const;`
Initialize a `vertex_cell` iterator to loop over all vertices.

`void begin(FEL_cell_iter* ci) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_vertex_cell_iter* vi, int k, ...)  const;`
`int begin(FEL_cell_iter* ci, int k, ...)  const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter* ci) const;`
Initialize an iterator to test if at end.

`bool is_mesh() const;`
Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Typedefs:

FEL_gradient_field1<FEL_vector3f, float>
FEL_gradient_of_float_field1;

Class: `FEL_gradient_field2<TO, FROM>`

Description:

Class supporting second-order accurate estimation of gradients.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_differential_operator_field` →
`FEL_differential_operator_field2` →
`FEL_gradient_field2`.

Public Member Functions:

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

`FEL_mesh_ptr get_mesh() const;`
Retrieve a pointer to the mesh associated with this field.

`int card(int d) const;`
Get the number of cells of specified dimension d.

`int coordinates_at_cell(const FEL_cell&, FEL_vector3f [])  const;`
Get physical-space coordinates of specified cell.

`void begin(FEL_vertex_cell_iter* vi) const;`
Initialize a `vertex_cell` iterator to loop over all vertices.

`void begin(FEL_cell_iter* ci) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_vertex_cell_iter* vi, int k, ...)  const;`
`int begin(FEL_cell_iter* ci, int k, ...)  const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter* ci) const;`
Initialize an iterator to test if at end.

`bool is_mesh() const;`
Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Typedefs:

FEL_gradient_field2<FEL_vector3f, float>
FEL_gradient_of_float_field2;

Class: `FEL_gradient_interpolator<TO, FROM>`

Description:

This class calulates gradients by first-order accurate methods, getting the necessary derivatives by analytic differentiation of either isoparametric or physical space interpolation polynomials.

Inheritance Hierarchy:

`FEL_derivative_interpolator` → `FEL_gradient_interpolator`.

Public Member Functions:

`int interpolate(const FEL_interpolant*, const FROM&, const FROM&, const FROM&, const FROM&, TOO[])=0;`

Interface for derivative interpolation on a tetrahedron.

`int interpolate(const FEL_interpolant*, const FEL_mesh*, const FEL_cell&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, TOO[])=0;`

Interface for derivative interpolation on a hexahedron.

`// the;`
`according to interpolation scheme specified by the ///`
`FEL_interpolant, and put answer in;`

Calculate the gradient at an interior point (physical coords, in `FEL_vector3f`) of a hexahedron, whose node values are specified by

`int interpolate(const FEL_vector3f&, const FEL_interpolant*, const FROM&, const FROM&, const FROM&, const FROM&, TO*);`

Calculate the gradient at an interior point (physical coords, in `FEL_vector3f`) of a tetrahedron, whose node values are specified by the `FROM`'s, according to interpolation scheme specified by the `FEL_interpolant`, and put answer in `TO*`.

`int interpolate(const FEL_vector3f&, const FEL_interpolant*, const int&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, TO*);`

Calculate the gradient at an interior point (physical coords, in `FEL_vector3f`) of a hexahedron, whose node values are specified by

`int interpolate(const FEL_interpolant*, const FROM&, const FROM&, const FROM&, const FROM&, TO[]);`

Calculate the gradient at all vertices of a tetrahedron, according to interpolation scheme specified by `FEL_interpolant`, and put answers in `TO[]`.

`int interpolate(const FEL_interpolant*, const FEL_mesh*, const FEL_cell&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, const FROM&, TO[]);`

Calculate the gradient at all vertices of a hexahedron, according to interpolation scheme specified by `FEL_interpolant`, and put answers in `TO[]`.

Class: `FEL_hash_cached_field<T>`

Description:

A wrapper field which caches node query results on another field, and satisfies repeated requests from its cache. FEL_hash_cached_field uses a dynamically created hash table for its storage needs (compare with FEL_cached_field).

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_hash_cached_field`.

Public Member Functions:

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```

Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f []) **const;**
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_time_steps() const;
Get number of timesteps associated with this field.

FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());
Construct a new core field where each node has been eagerly evaluated.

int get_min_max(T*, T*, const FEL_time& = FEL_time());
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Typedefs:

FEL_hash_cached_field<float>

FEL_hash_cached_float_field;

FEL_hash_cached_field<FEL_vector3f>

FEL_hash_cached_vector3f_field;

Class: `FEL_hash_dict<KEY, VALUE>`

Description:

FEL_hash_dict represents a dictionary data structure using a hash table.

Public Member Functions:

```
FEL_hash_dict(int n);
```

Initialize a hash table of size n

```
bool lookup(const KEY&, VALUE*);
bool lookup(const KEY&, KEY*, VALUE*);
void put(const KEY&, const VALUE&);
void del(const KEY&);
```

Access table values

```
int card();
```

Return the number of items in the table

Class: FEL_hexahedral_isoparametric_interpolant

Description:

Supports isoparametric interpolation on hexahedra.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_interpolant →
FEL_isoparametric_interpolant →
FEL_isoparametric_3D_interpolant →
FEL_hexahedral_isoparametric_interpolant.

Public Member Functions:

FEL_hexahedral_isoparametric_interpolant(const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&);

Construct the interpolant with the eight vertices of a hexahedron.

const char* get_name() const;
void set_name(const char*);

Access the name of this object.

bool ok() const;

Indicates if interpolant was successfully constructed. Workaround for lack of exceptions on many systems.

int physical_to_computational_coords(const FEL_vector3d&, double&,double&,double&, double tol=1e-6);

Converts physical coordinates (input as FEL_vector3d&) into local computational coordinates, returned in the double&'s. The conversion uses Newton-Raphson iteration, with tolerated error < tol. Returns 0 if no convergence.

int inverse_jacobian(FEL_matrix33d&, double, double, double);

Calculate the inverse of the Jacobian matrix at the location specified by the doubles, and return it in the FEL_matrix33d&.

int inverse_jacobian_t(FEL_matrix33d&, double, double, double);

Calculate the inverse of the transpose of the Jacobian matrix at the location specified by the doubles, and return it in the FEL_matrix33d&.

bool is_hexahedral_isoparametric() const;

Return true.

FEL_interpolation_enum get_interpolation() const;

Interface for retrieving interpolation mode.

bool is_mesh() const;

`bool is_field() const;`

Provide run-time type determination for a few key FEL types

Class: `FEL_hexahedral_physical_interpolant`

Description:

Support for physical space interpolation on hexahedra.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_interpolant` →
`FEL_physical_interpolant` →
`FEL_hexahedral_physical_interpolant`.

Public Member Functions:

`FEL_hexahedral_physical_interpolant(const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&);`

Construct the interpolant with the eight vertices of a hexahedron.

`const char* get_name() const;`
`void set_name(const char*);`

Access the name of this object.

`bool ok() const;`

Indicates if interpolant was successfully constructed. Workaround for lack of exceptions on many systems.

`bool is_hexahedral_isoparametric() const;`

Indicates whether calling object is an `FEL_hexahedral_isoparaemtric_interpolant`.

`FEL_interpolation_enum get_interpolation() const;`

Interface for retrieving interpolation mode.

`bool is_mesh() const;`
`bool is_field() const;`

Provide run-time type determination for a few key FEL types

Class: `FEL_interpolant`

Description:
Abstract base class for all interpolants.

Inheritance Hierarchy:
`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_interpolant`.

Public Member Functions:

`const char* get_name() const;`
`void set_name(const char*);`
Access the name of this object.

`bool ok() const;`
Indicates if interpolant was successfully constructed. Workaround for lack of exceptions on many systems.

`bool is_hexahedral_isoparametric() const;`
Indicates whether calling object is an `FEL_hexahedral_isoparaemtric_interpolant`.

`FEL_interpolation_enum get_interpolation() const;`
Interface for retrieving interpolation mode.

`bool is_mesh() const;`
`bool is_field() const;`
Provide run-time type determination for a few key FEL types

Class: `FEL_interpolate_then_map_derived_field1<TO, FROM>`

Description:

Derived field built on one component field, which performs interpolation prior to derivation.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_derived_field` →
`FEL_map_then_interpolate_derived_field1` →
`FEL_interpolate_then_map_derived_field1`.

Public Member Functions:

```
FEL_interpolate_then_map_derived_field1(FEL_pointer<
  FEL_typed_field<FROM> >, int (*)(const
  FEL_solution_globals&, const FROM*, void*, TO*),
  void*, char* = "interpolate_then_map_derived_field1");
```
Construct a derived field with one component field.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```
Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```
Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```
Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```
Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
```

bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f []) **const;**
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) **const;**
int begin(FEL_cell_iter* ci, int k, ...) **const;**
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = **FEL_ZONE_UNDEFINED) const;**
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, **FEL_vector3f*) const;**
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, **FEL_time*) const;**
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`

Get number of timesteps associated with this field.

Class: `FEL_interpolate_then_map_derived_field2<TO, FROM1, FROM2>`

Description:

Derived field built on two component fields, which performs interpolation prior to derivation.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_derived_field` →
`FEL_map_then_interpolate_derived_field2` →
`FEL_interpolate_then_map_derived_field2`.

Public Member Functions:

```
FEL_interpolate_then_map_derived_field2(FEL_pointer<
  FEL_typed_field<FROM1> >, FEL_pointer<
  FEL_typed_field<FROM2> >, int (*)(const
  FEL_solution_globals&, const FROM1*, const
  FROM2*, void*, TO*), void*, char* = "interpo-
  late_then_map_derived_field2");
```

Construct a derived field with two component fields.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```
Return true if calling object is an instance of corresponding class.

```
bool varies_with_time() const;
```
Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```
Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```
Get the number of cells of specified dimension d.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f
  []) const;
```
Get physical-space coordinates of specified cell.

```
void begin(FEL_vertex_cell_iter* vi) const;
```
Initialize a `vertex_cell` iterator to loop over all vertices.

```
void begin(FEL_cell_iter* ci) const;
```
Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_vertex_cell_iter* vi, int k, ...)  const;
int begin(FEL_cell_iter* ci, int k, ...)  const;
```
Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter* ci) const;
```
Initialize an iterator to test if at end.

```
bool is_mesh() const;
```
Provide run-time type determination for a few key FEL types

```
bool is_field() const;
```
Return true if calling object is an instance of corresponding class.

```
int get_n_zones() const;
```
Get number of zones in this field's mesh.

```
void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Set an option on this field, or its mesh.

```
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
```
Get physical-space coordinates of specified cell.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Class: FEL_interpolate_then_map_derived_field3<TO, FROM1, FROM2, FROM3>

Description:

Derived field built on three component fields, which performs interpolation prior to derivation.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_field →
FEL_typed_field →
FEL_derived_field →
FEL_map_then_interpolate_derived_field3 →
FEL_interpolate_then_map_derived_field3.

Public Member Functions:

```
FEL_interpolate_then_map_derived_field3(FEL_pointer<
  FEL_typed_field<FROM1> >, FEL_pointer<
  FEL_typed_field<FROM2> >, FEL_pointer<
  FEL_typed_field<FROM3> >, int (*)(const
  FEL_solution_globals&, const FROM1*, const FROM2*,
  const FROM3*, void*, TO*), void*, char* = "interpo-
  late_then_map_derived_field3");
```

Construct a derived field with three component fields.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
Get the minimum and maximum values of the field (undefined for non-scalar fields).

bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f []) **const;**
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) **const;**
int begin(FEL_cell_iter* ci, int k, ...) **const;**
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = **FEL_ZONE_UNDEFINED) const;**
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, **FEL_vector3f*) const;**
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`

Change from one time representation to another.

`bool is_time_series_field() const;`

Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`

Get number of timesteps associated with this field.

Class: `FEL_interpolator<T>`

Description:

Class providing interpolation functions on various cell types.

Public Member Functions:

`FEL_interpolator();`

Construct a generic interpolator.

`int interpolate(const FEL_vector3f&, const FEL_interpolant*, const T&, const T&, const T&, const T&, T*);`
`int interpolate(const FEL_vector3f&, const FEL_interpolant*, const T&, const T&, const T&, const T&, const T&, T*);`
`int interpolate(const FEL_vector3f&, const FEL_interpolant*, const T&, const T&, const T&, const T&, const T&, const T&, T*);`
`int interpolate(const FEL_vector3f&, const FEL_interpolant*, const int&, const T&, const T&, const T&, const T&, const T&, const T&, const T&, const T&, T*);`

Interpolate on a cell with number of `(const T)` nodes, according to method specified in the provided `FEL_interpolant`.

`int bilinear_interpolate(double, double, const T&, const T&, const T&, const T&, T*);`

Bilinear interpolation on an orthonormal quadrilateral, with node values specified by the incoming `T`'s, at location indicated by the incoming pair of `double`s.

`int trilinear_interpolate(double, double, double, const T&, const T&, const T&, const T&, const T&, const T&, const T&, const T&, const T&, T*);`

Trilinear interpolation on an orthonormal hexahedron, with node values specified by the incoming `T`'s, at location indicated by the incoming triplet of `double`s.

Class: `FEL_irregular_axis`

Description:

Inheritance Hierarchy:

`FEL_axis` →
`FEL_irregular_axis`.

Public Member Functions:

Class: `FEL_isoparametric_2D_interpolant`

Description:

Base class for two-dimensional isoparametric interpolants.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_interpolant` →
`FEL_isoparametric_interpolant` →
`FEL_isoparametric_2D_interpolant`.

Public Member Functions:

`const char* get_name() const;`
`void set_name(const char*);`

Access the name of this object.

`bool ok() const;`

Indicates if interpolant was successfully constructed. Workaround for lack of exceptions on many systems.

`int physical_to_computational_coords(const FEL_vector2d&, double&,double&, double tol=1e-6);`

Converts physical coordinates (input as `FEL_vector2d&`) into local computational coordinates, returned in the `double&`'s. The conversion uses Newton-Raphson iteration, with tolerated error `< tol`. Returns 0 if no convergence.

`bool is_hexahedral_isoparametric() const;`

Indicates whether calling object is an `FEL_hexahedral_isoparaemtric_interpolant.`

`FEL_interpolation_enum get_interpolation() const;`

Interface for retrieving interpolation mode.

`bool is_mesh() const;`
`bool is_field() const;`

Provide run-time type determination for a few key FEL types

Class: `FEL_isoparametric_3D_interpolant`

Description:

Base class for three-dimensional isoparametric interpolants.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_interpolant` →
`FEL_isoparametric_interpolant` →
`FEL_isoparametric_3D_interpolant`.

Public Member Functions:

`const char* get_name() const;`
`void set_name(const char*);`

Access the name of this object.

`bool ok() const;`

Indicates if interpolant was successfully constructed. Workaround for lack of exceptions on many systems.

`int physical_to_computational_coords(const FEL_vector3d&, double&,double&,double&, double tol=1e-6);`

Converts physical coordinates (input as `FEL_vector3d&`) into local computational coordinates, returned in the `double&`'s. The conversion uses Newton-Raphson iteration, with tolerated error `< tol`. Returns 0 if no convergence.

`bool is_hexahedral_isoparametric() const;`

Indicates whether calling object is an `FEL_hexahedral_isoparaemtric_interpolant.`

`FEL_interpolation_enum get_interpolation() const;`

Interface for retrieving interpolation mode.

`bool is_mesh() const;`
`bool is_field() const;`

Provide run-time type determination for a few key FEL types

Class: `FEL_isoparametric_interpolant`

Description:
Base class for all isoparametric interpolants.

Inheritance Hierarchy:
`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_interpolant` →
`FEL_isoparametric_interpolant`.

Public Member Functions:

`const char* get_name() const;`
`void set_name(const char*);`
Access the name of this object.

`bool ok() const;`
Indicates if interpolant was successfully constructed. Workaround for lack of exceptions on many systems.

`bool is_hexahedral_isoparametric() const;`
Indicates whether calling object is an `FEL_hexahedral_isoparaemtric_interpolant`.

`FEL_interpolation_enum get_interpolation() const;`
Interface for retrieving interpolation mode.

`bool is_mesh() const;`
`bool is_field() const;`

Class: `FEL_magnitude_field<TO, FROM>`

Description:

Derived field returning magnitude of its component field – typically the length (Euclidean norm) of a vector.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_derived_field` →
`FEL_map_then_interpolate_derived_field1` →
`FEL_magnitude_field`.

Public Member Functions:

```
FEL_magnitude_field(FEL_pointer< FEL_typed_field<FROM> >
  f, char* nm = "magnitude_field");
```
Construct this built-in derived field.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```
Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```
Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```
Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```
Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
```

bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f [ ]) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Typedefs:

`FEL_magnitude_field<float, FEL_vector3f>`
`FEL_magnitude_of_vector3f_field;`

Class: `FEL_map_then_interpolate_derived_field1<TO, FROM>`

Description:

Derived field built on one component field, which performs derivation prior to interpolation.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_derived_field` →
`FEL_map_then_interpolate_derived_field1`.

Public Member Functions:

```
FEL_map_then_interpolate_derived_field1(FEL_pointer<
  FEL_typed_field<FROM> >, int (*)(const
  FEL_solution_globals&, const FROM*, void*, TO*),
  void*, char* = "map_then_interpolate_derived_field1");
```
Construct a derived field with one component field.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```
Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```
Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```
Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```
Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
```

bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f [ ]) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`

Get number of timesteps associated with this field.

Class: `FEL_map_then_interpolate_derived_field2<TO, FROM1, FROM2>`

Description:

Derived field built on two component fields, which performs derivation prior to interpolation.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_derived_field` →
`FEL_map_then_interpolate_derived_field2`.

Public Member Functions:

```
FEL_map_then_interpolate_derived_field2(FEL_pointer<
  FEL_typed_field<FROM1> >, FEL_pointer<
  FEL_typed_field<FROM2> >, int (*)(const
  FEL_solution_globals&, const FROM1*,
  const FROM2*, void*, TO*), void*, char* =
  "map_then_interpolate_derived_field2");
```

Construct a derived field with two component fields.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f [ ]) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;

Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`

Get number of timesteps associated with this field.

Class: `FEL_map_then_interpolate_derived_field3<TO, FROM1, FROM2, FROM3>`

Description:

Derived field built on three component fields, which performs derivation prior to interpolation.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_derived_field` →
`FEL_map_then_interpolate_derived_field3`.

Public Member Functions:

```
FEL_map_then_interpolate_derived_field3(FEL_pointer<
  FEL_typed_field<FROM1> >, FEL_pointer<
  FEL_typed_field<FROM2> >, FEL_pointer<
  FEL_typed_field<FROM3> >, int (*)(const
  FEL_solution_globals&, const FROM1*, const
  FROM2*, const FROM3*, void*, TO*), void*, char* =
  "map_then_interpolate_derived_field3");
```

Construct a derived field with three component fields.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```
Return true if calling object is an instance of corresponding class.

```
bool varies_with_time() const;
```
Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```
Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```
Get the number of cells of specified dimension d.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f
  []) const;
```
Get physical-space coordinates of specified cell.

```
void begin(FEL_vertex_cell_iter* vi) const;
```
Initialize a `vertex_cell` iterator to loop over all vertices.

```
void begin(FEL_cell_iter* ci) const;
```
Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_vertex_cell_iter* vi, int k, ...)  const;
int begin(FEL_cell_iter* ci, int k, ...)  const;
```
Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter* ci) const;
```
Initialize an iterator to test if at end.

```
bool is_mesh() const;
```
Provide run-time type determination for a few key FEL types

```
bool is_field() const;
```
Return true if calling object is an instance of corresponding class.

```
int get_n_zones() const;
```
Get number of zones in this field's mesh.

```
void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Set an option on this field, or its mesh.

```
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
```
Get physical-space coordinates of specified cell.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Class: FEL_matrix22<T>

Description:

A 2 by 2 matrix.

Public Member Functions:

```
FEL_matrix22();
FEL_matrix22(const T& d00, const T& d01, const T& d10,
  const T& d11);
```
Initialize.

```
friend ostream& operator<<(ostream& strm, const
  FEL_matrix22<T>& m);
```
Write to ostream.

```
FEL_matrix22(FEL_vector2<T>&, FEL_vector2<T>&, int);
void set(const T& d00, const T& d01, const T& d10,
  const T& d11);
FEL_vector2<T>& operator[](int i);
const FEL_vector2<T>& operator[](int i) const;
```
Access the matrix values.

```
FEL_matrix22& operator*=(double f);
```
Do math.

```
friend T FEL_determinant(const FEL_matrix22<T>& m);
```
Return determinant.

```
friend int FEL_invert(const FEL_matrix22<T>& m,
  FEL_matrix22<T>* res);
```
Analytically invert matrix.

Typedefs:

FEL_matrix22<double>
FEL_matrix22d;

FEL_matrix22<float>
FEL_matrix22f;

Class: FEL_matrix33<T>

Description:

A 3 by 3 matrix.

Public Member Functions:

```
FEL_matrix33();
FEL_matrix33(const T& d00, const T& d01, const T& d02,
  const T& d10, const T& d11, const T& d12, const T&
  d20, const T& d21, const T& d22);
FEL_matrix33(const FEL_vector3<T>&, const
  FEL_vector3<T>&, const FEL_vector3<T>&, int);
```

Initialize.

```
friend ostream& operator<<(ostream& strm, const
  FEL_matrix33<T>& m);
```

Write to an ostream.

```
void set(const T& d00, const T& d01, const T& d02,
  const T& d10, const T& d11, const T& d12, const T&
  d20, const T& d21, const T& d22);
FEL_vector3<T>& operator[](int i);
const FEL_vector3<T>& operator[](int i) const;
```

Access matrix elements.

```
FEL_matrix33<T>& operator*=(const T& f);
friend FEL_matrix33<T> operator+(const FEL_matrix33<T>&
  lhs, const FEL_matrix33<T>& rhs);
friend FEL_matrix33<T> operator*(double lhs, const
  FEL_matrix33<T>& rhs);
```

Do math.

```
friend T FEL_determinant(const FEL_matrix33<T>& m);
```

Return determinant.

```
// friend int FEL_safe_invert(const FEL_matrix33<T>&,
  FEL_matrix33<T>*);
```

Invert using SVD.

Typedefs:

FEL_matrix33<double>

FEL_matrix33d;

FEL_matrix33<float>

FEL_matrix33f;

Class: FEL_matrix44<T>

Description:

A 4 by 4 matrix.

Public Member Functions:

```
FEL_matrix44();
FEL_matrix44(const T&, const T&, const T&, const T&,
  const T&, const T&, const T&, const T&, const T&,
  const T&, const T&, const T&, const T&, const T&,
  const T&, const T&);
```
Initialize.

```
friend ostream& operator<<(ostream& strm, const
  FEL_matrix44<T>& m);
```
Write to an ostream.

```
void set(const T& d00, const T& d01, const T& d02,
  const T& d03, const T& d10, const T& d11, const T&
  d12, const T& d13, const T& d20, const T& d21, const
  T& d22, const T& d23, const T& d30, const T& d31,
  const T& d32, const T& d33);
FEL_vector4<T>& operator[](int i);
const FEL_vector4<T>& operator[](int i) const;
```
Access data members.

```
// friend int FEL_invert(const FEL_matrix44<T>&,
  FEL_matrix44<T>*);
```
Invert the matrix.

```
// friend int FEL_safe_invert(const FEL_matrix44<T>&,
  FEL_matrix44<T>*);
```
Invert the matrix using SVD.

Typedefs:

FEL_matrix44<double>
FEL_matrix44d;

FEL_matrix44<float>
FEL_matrix44f;

Class: `FEL_matrix55<T>`

Description:

A 5 by 5 matrix.

Public Member Functions:

`FEL_matrix55();`

Initialize.

`friend ostream& operator<<(ostream&, const FEL_matrix55<T>&);`

Write to an ostream.

`//friend int FEL_invert(const FEL_matrix55<T>&, FEL_matrix55<T>*);`

Invert the matrix.

Typedefs:

`FEL_matrix55<double>`

`FEL_matrix55d;`

`FEL_matrix55<float>`

`FEL_matrix55f;`

Class: FEL_matrix66<T>

Description:

A 6 by 6 matrix.

Public Member Functions:

FEL_matrix66();

Initialize.

friend ostream& operator<<(ostream&, const FEL_matrix66<T>&);

Write to an ostream.

//friend int FEL_invert(const FEL_matrix66<T>&, FEL_matrix66<T>*);

Invert the matrix.

Typedefs:

FEL_matrix66<double>

FEL_matrix66d;

FEL_matrix66<float>

FEL_matrix66f;

Class: `FEL_matrix88<T>`

Description:

A 8 by 8 matrix.

Public Member Functions:

`FEL_matrix88();`

Initialize.

`friend ostream& operator<<(ostream& strm, const FEL_matrix88<T>& m);`

Write to an ostream.

`FEL_vector8<T>& operator[](int i);`
`const FEL_vector8<T>& operator[](int i) const;`

Access matrix data.

`// friend int FEL_invert(const FEL_matrix88<T>&, FEL_matrix88<T>*);`

Invert the matrix.

`//friend int FEL_safe_invert(const FEL_matrix88<T>&, FEL_matrix88<T>*);`

Invert the matrix using SVD.

Typedefs:

`FEL_matrix88<double>`

`FEL_matrix88d;`

`FEL_matrix88<float>`

`FEL_matrix88f;`

Class: `FEL_mesh`

Description:

The class FEL_mesh declares the interface inherited by all mesh classes.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_mesh_ptr get_zone(int) const;
```
Return a pointer to the zone specified by the argument.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
bool is_structured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```

Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
```

int coordinates_and_iblank_at_cell(const FEL_cell&,
FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
FEL_vertex_cell&, FEL_vector3f_and_int*) const;
Access coordinates and IBLANK.

int convert_time(const FEL_time&, FEL_time_representation_enum,
FEL_time*) const;
Convert time representation.

void begin(FEL_vertex_cell_iter*) const;
Initialize a `vertex_cell` iterator to loop over all the vertices.

void begin(FEL_cell_iter*) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_cell_iter*, int, ...) const;
int begin(FEL_vertex_cell_iter*, int, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter*) const;
Initialize an iterator to test if at end.

bool is_unstructured_mesh() const;
Provide run-time type determination for a few mesh subclasses.

Class: `FEL_mesh_as_field<T>`

Description:

This field type in effect creates a vector field whose entry at each node is just the position vector of the node, as given by the mesh. Thus this field type constitutes an *adaptor*, allowing one to access a mesh using the field interface.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_mesh_as_field`.

Public Member Functions:

```
FEL_mesh_as_field(FEL_mesh_ptr, char* = "mesh_as_field");
```
Construct a field interface to a mesh.

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```
Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```
Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```
Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```
Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f [ ]) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_time_steps() const;
Get number of timesteps associated with this field.

FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());
Construct a new core field where each node has been eagerly evaluated.

int get_min_max(T*, T*, const FEL_time& = FEL_time());
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Typedefs:

FEL_mesh_as_field<FEL_vector3f_and_int>
FEL_mesh_as_vector3f_and_int_field;

FEL_mesh_as_field<FEL_vector3f>
FEL_mesh_as_vector3f_field;

Class: `FEL_multi_field<T>`

Description:

This field type is basically a container class capable of managing multiple fields. It is used primarily for supporting transformed fields.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_multi_field`.

Public Member Functions:

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```

Get the number of cells of specified dimension d.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f
  []) const;
```

Get physical-space coordinates of specified cell.

`void begin(FEL_vertex_cell_iter* vi) const;`
Initialize a `vertex_cell` iterator to loop over all vertices.

`void begin(FEL_cell_iter* ci) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_vertex_cell_iter* vi, int k, ...) const;`
`int begin(FEL_cell_iter* ci, int k, ...) const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter* ci) const;`
Initialize an iterator to test if at end.

`bool is_mesh() const;`
Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

`FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());`
Construct a new core field where each node has been eagerly evaluated.

`int get_min_max(T*, T*, const FEL_time& = FEL_time());`
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Typedefs:

`FEL_multi_field<float>`
`FEL_multi_float_field;`

```
FEL_multi_field<FEL_plot3d_q>
    FEL_multi_plot3d_q_field;
FEL_multi_field<FEL_vector3f>
    FEL_multi_vector3f_field;
```

Class: FEL_multi_mesh

Description:

FEL_multi_mesh represents multi-zone meshes.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_mesh →
FEL_multi_mesh.

Public Member Functions:

```
FEL_multi_mesh(int, FEL_mesh_ptr*, const char* nm =
  "FEL_multi_mesh");
```
Construct a multi_mesh.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
bool is_structured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
```

int combined_iblank_at_cell(const FEL_cell&, int*) const;
int coordinates_and_iblank_at_cell(const FEL_cell&, FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f_and_int*) const;
Access coordinates and IBLANK.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Convert time representation.

void begin(FEL_vertex_cell_iter*) const;
Initialize a `vertex_cell` iterator to loop over all the vertices.

void begin(FEL_cell_iter*) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_cell_iter*, int, ...) const;
int begin(FEL_vertex_cell_iter*, int, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter*) const;
Initialize an iterator to test if at end.

bool is_unstructured_mesh() const;
Provide run-time type determination for a few mesh subclasses.

Class: `FEL_mutex_reference_counted_object`

Description:

Objects that are reference counted inherit from this class. The reference counting has critical section protection, thus it is safe to use in multi-threaded scenarios.

Inheritance Hierarchy:

`FEL_reference_counted_object` → `FEL_mutex_reference_counted_object`.

Public Member Functions:

`const char* get_name() const;`
`void set_name(const char*);`

Access the name of this object.

`bool is_mesh() const;`
`bool is_field() const;`

Provide run-time type determination for a few key FEL types

Class: `FEL_nearest_neighbor_interpolant`

Description:

Interpolant supporting nearest-neighbor ”interpolation”.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_interpolant` →
`FEL_nearest_neighbor_interpolant`.

Public Member Functions:

`FEL_nearest_neighbor_interpolant(int, FEL_vector3f[]);`

Initializes the interpolant with `int` nearest neighbors, whose coordinates are stored in the `FEL_vector3f[]`.

`const char* get_name() const;`
`void set_name(const char*);`

Access the name of this object.

`bool ok() const;`

Indicates if interpolant was successfully constructed. Workaround for lack of exceptions on many systems.

`int get_nearest(const FEL_vector3f&);`

Returns index of neighbor nearest input point `FEL_vector3f&`.

`bool is_hexahedral_isoparametric() const;`

Indicates whether calling object is an `FEL_hexahedral_isoparaemtric_interpolant`.

`FEL_interpolation_enum get_interpolation() const;`

Returns interpolation mode of this object, namely `FEL_NEAREST_NEIGHBOR_INTERPOLATION`.

`bool is_mesh() const;`
`bool is_field() const;`

Provide run-time type determination for a few key FEL types

Class: FEL_negate_field<TO, FROM>

Description:

Derived field returning additive inverse of its component field.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_field →
FEL_typed_field →
FEL_derived_field →
FEL_map_then_interpolate_derived_field1 →
FEL_negate_field.

Public Member Functions:

```
FEL_negate_field(FEL_pointer< FEL_typed_field<FROM> > f,
  char* nm = "negate_field");
```
Construct this built-in derived field.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```
Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```
Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```
Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```
Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
```

bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f []) **const;**
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) **const;**
int begin(FEL_cell_iter* ci, int k, ...) **const;**
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = **FEL_ZONE_UNDEFINED) const;**
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, **FEL_vector3f*) const;**
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, **FEL_time*) const;**
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_time_steps() const;

Get number of timesteps associated with this field.

Typedefs:

`FEL_negate_field<float, float>`
`FEL_negate_of_float_field;`

`FEL_negate_field<FEL_vector3f, FEL_vector3f>`
`FEL_negate_of_vector3f_field;`

Class: `FEL_neighbor`

Description:

FEL_neighbor encodes an identity number, a 3-cell type and a face number in an unsigned int. FEL_neighbor is used by FEL_unstructured mesh to store the information needed to get from a 3-cell to one of its neighbors.

Public Member Functions:

`FEL_neighbor(int i, int t, int f);`
`FEL_neighbor();`
Initialize.

`friend ostream& operator<<(ostream&, const FEL_neighbor&);`
Write to ostream.

`int type() const;`
Return the 3-cell type.

`int face() const;`
Return the face number.

`int id() const;`
Return the 3-cell identity number.

`bool defined();`
Return true values have been defined.

Class: `FEL_null_axis`

Description:

Inheritance Hierarchy:

`FEL_axis` →
`FEL_null_axis`.

Public Member Functions:

Class: `FEL_paged_curvilinear_mesh_xyz`

Description:

The FEL_paged_curvilinear_mesh_xyz class implements paged curvilinear meshes that don't have iblanks. Instances are created with FEL_plot3d_read_paged_mesh.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_single_mesh` →
`FEL_structured_mesh` →
`FEL_curvilinear_mesh` →
`FEL_paged_curvilinear_mesh_xyz`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
int get_n_zones() const;
```
Return 1 for any subclass of `FEL_single_mesh`.

```
FEL_mesh_ptr get_zone(int) const;
```
Return the mesh itself the argument is 0, and NULL otherwise.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
int jacobian_at_vertex_cell(const FEL_vertex_cell&,
  FEL_matrix33f*) const;
```

Request Jacobian matrix at given vertex.

```
int contravariant_phys_to_comp_vector(const
  FEL_vertex_cell&, const FEL_vector3f&, FEL_vector3f*)
  const;
int contravariant_phys_to_comp_vector(const
  FEL_structured_pos&, const FEL_vector3f&,
  FEL_vector3f*) const;
```

Convert a contravariant vector in physical space to its equivalent in computational space.

```
bool is_structured_mesh() const;
```

Return true.

```
void set(const FEL_set_keyword_enum, int);
```

Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```

Get a mesh parameter.

```
int card(int k) const;
```

Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```

Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```

Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```

Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```

Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```

Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```

Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```

Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
```

```
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```

Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: `FEL_paged_curvilinear_mesh_xyzi`

Description:

The FEL_paged_curvilinear_mesh_xyzi class implements paged curvilinear meshes that have iblanks. Instances are are created with FEL_plot3d_read_paged_mesh.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_single_mesh` →
`FEL_structured_mesh` →
`FEL_curvilinear_mesh` →
`FEL_paged_curvilinear_mesh_xyz` →
`FEL_paged_curvilinear_mesh_xyzi`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
int get_n_zones() const;
```
Return 1 for any subclass of `FEL_single_mesh`.

```
FEL_mesh_ptr get_zone(int) const;
```
Return the mesh itself the argument is 0, and NULL otherwise.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
int jacobian_at_vertex_cell(const FEL_vertex_cell&,
  FEL_matrix33f*) const;
```
Request Jacobian matrix at given vertex.

```
int contravariant_phys_to_comp_vector(const
  FEL_vertex_cell&, const FEL_vector3f&, FEL_vector3f*)
  const;
int contravariant_phys_to_comp_vector(const
  FEL_structured_pos&, const FEL_vector3f&,
  FEL_vector3f*) const;
```
Convert a contravariant vector in physical space to its equivalent in computational space.

```
bool is_structured_mesh() const;
```
Return true.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```

Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: `FEL_paged_field<T>`

Description:

The FEL_paged_field class implements paged fields. Instances should be created with the file reader functions (FEL_plot3d_read_paged_*).

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_paged_field`.

Public Member Functions:

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```

Get the number of cells of specified dimension d.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f
  []) const;
```

Get physical-space coordinates of specified cell.

`void begin(FEL_vertex_cell_iter* vi) const;`
Initialize a `vertex_cell` iterator to loop over all vertices.

`void begin(FEL_cell_iter* ci) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_vertex_cell_iter* vi, int k, ...) const;`
`int begin(FEL_cell_iter* ci, int k, ...) const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter* ci) const;`
Initialize an iterator to test if at end.

`bool is_mesh() const;`
Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

`FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());`
Construct a new core field where each node has been eagerly evaluated.

`int get_min_max(T*, T*, const FEL_time& = FEL_time());`
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Typedefs:

`FEL_paged_field<float>`
`FEL_paged_float_field;`

FEL_paged_field<FEL_plot3d_density_momentum>
FEL_paged_plot3d_density_momentum_field;

FEL_paged_field<FEL_plot3d_q>
FEL_paged_plot3d_q_field;

FEL_paged_field<FEL_vector3f>
FEL_paged_vector3f_field;

Class: `FEL_paged_field_common`

Description:

The FEL_paged_field_common class is part of the implementation of paged fields. All members are private.

Public Member Functions:

Class: `FEL_paged_file`

Description:

The class FEL_paged_file provides much of the implementation for paged meshes and fields. Most FEL users should not use this class; use paged meshes and fields instead. Because this is an implementation class, the interface is subject to revision or possibly to being revoked by making all members private.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_paged_file`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool has_iblank() const;
FEL_vector3i get_dim(int z) const;
int get_dim(int z, int i) const;
float get_fs_mach(int z) const;
float get_reynolds(int z) const;
float get_alpha(int z) const;
float get_integration(int z) const;
float get_param_max(int z, int p) const;
float get_param_min(int z, int p) const;
int get_iblank_max(int z) const;
int get_iblank_min(int z) const;
File_type get_type() const;
const char* get_type_str() const;
const char* get_filename() const;
float get_version() const;
int get_n_fields() const;
```

Inquiry functions

```
void get_vert(int zone, const FEL_vector3i& v,
  Param_offset fstrt, Param_length fcnt, void* out);
void get_hex_cell(int zone, const FEL_vector3i&
  v, Param_offset fstrt, Param_length fcnt, int
  data_stride, void* out);
void get_hex_subcell(int zone, const FEL_vector3i& v,
  const int* vert_id, int n_verts, Param_offset fstrt,
  Param_length fcnt, int data_stride, void* out);
void set_page_priority(int z, int priority);
```

Read functions

```
int write_header();
int write_block(void* p, int len);
int write_blocks(int count, void* p, int* len);
```

```
void set_little_endian(bool le);
void set_dim(int z, FEL_vector3i dd);
void set_dim(int z, int i, int j, int k);
void set_fs_mach(int z, float v);
void set_reynolds(int z, float v);
void set_alpha(int z, float v);
void set_integration(int z, float v);
void set_param_max(int z, int p, float v);
void set_param_min(int z, int p, float v);
void set_iblank_max(int z, long v);
void set_iblank_min(int z, long v);
```

Write functions

```
bool is_mesh() const;
```

Inquiry functions

```
bool is_field() const;
```

Provide run-time type determination for a few key FEL types

```
int get_n_zones() const;
```

Inquiry functions

Class: FEL_phys_pos

Description:

The class FEL_phys_pos represents a point in physical space. FEL_phys_pos also contains an FEL_time data member.

Inheritance Hierarchy:

FEL_vector3f → FEL_phys_pos.

Public Member Functions:

```
FEL_phys_pos();
FEL_phys_pos(const FEL_vector3f& p);
FEL_phys_pos(const float& x, const float& y, const
  float& z);
```

Initialize an FEL_phys_pos.

```
FEL_time get_time() const;
float get_physical_time();
float get_computational_time();
int get_time_step();
void set_time(const FEL_time& t);
void set_physical_time(float t);
void set_computational_time(float t);
void set_time_step(int t);
void set_time_undefined();
```

Access the time representation.

```
friend ostream& operator<<(ostream&, const
  FEL_phys_pos&);
```

Output an FEL_phys_pos to an ostream.

```
FEL_phys_pos& operator+=(const FEL_vector3f& v);
FEL_phys_pos& operator-=(const FEL_vector3f& v);
```

Modify the spatial coordinates of the physical position.

```
friend bool operator==(const FEL_phys_pos& lhs, const
  FEL_phys_pos& rhs);
```

Test for equality, comparing both spatial coordinates and time.

Class: `FEL_physical_interpolant`

Description:

Base class for physical space interpolants.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_interpolant` →
`FEL_physical_interpolant`.

Public Member Functions:

`const char* get_name() const;`
`void set_name(const char*);`

Access the name of this object.

`bool ok() const;`

Indicates if interpolant was successfully constructed. Workaround for lack of exceptions on many systems.

`bool is_hexahedral_isoparametric() const;`

Indicates whether calling object is an `FEL_hexahedral_isoparaemtric_interpolant`.

`FEL_interpolation_enum get_interpolation() const;`

Interface for retrieving interpolation mode.

`bool is_mesh() const;`
`bool is_field() const;`

Provide run-time type determination for a few key FEL types

Class: `FEL_plot3d_density_momentum`

Description:

FEL_plot3d_density_momentum represents a subset of the PLOT3D solution vector containing just the components needed for velocity-related derived fields.

Public Member Functions:

```
FEL_plot3d_density_momentum();
FEL_plot3d_density_momentum(float d, FEL_vector3f m);
```
Initialize an object.

```
friend ostream& operator<<(ostream&, const
  FEL_plot3d_density_momentum&);
```
Write to an ostream.

```
friend FEL_plot3d_density_momentum operator-(const
  FEL_plot3d_density_momentum& dm);
friend FEL_plot3d_density_momentum opera-
  tor+(const FEL_plot3d_density_momentum& lhs, const
  FEL_plot3d_density_momentum& rhs);
friend FEL_plot3d_density_momentum operator-
  (const FEL_plot3d_density_momentum& lhs, const
  FEL_plot3d_density_momentum& rhs);
friend FEL_plot3d_density_momentum operator*(float lhs,
  const FEL_plot3d_density_momentum& rhs);
friend FEL_plot3d_density_momentum operator*(double lhs,
  const FEL_plot3d_density_momentum& rhs);
friend FEL_plot3d_density_momentum operator*(const
  FEL_plot3d_density_momentum& lhs, float rhs);
friend FEL_plot3d_density_momentum operator*(const
  FEL_plot3d_density_momentum& lhs, double rhs);
```
Do math.

```
friend bool operator==(const FEL_plot3d_density_momentum&
  lhs, const FEL_plot3d_density_momentum& rhs);
```
Test for equality.

Class: FEL_plot3d_field

Description:

Field "manager" which helps create and manage fields based on PLOT3D data files. This class can return over fifty predefined derived fields, in response to an enum'ed request, and can take care of any necessary file reading and primitive field caching. This is a virtual base class; use either FEL_q_plot3d_field, FEL_steady_plot3d_field, or FEL_fixed_interval_time_series_plot3d_field to create an instance.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_plot3d_field.

Public Member Functions:

```
FEL_mesh_ptr get_mesh();
FEL_plot3d_q_field_ptr get_q_field();
FEL_plot3d_density_momentum_field_ptr
  get_density_momentum_field();
FEL_vector3f_field_ptr get_momentum_field();
FEL_float_field_ptr get_density_field();
FEL_float_field_ptr get_energy_field();
```

Retrieve mesh, or PLOT3D primitive fields.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_float_field_ptr make_float_field(FEL_float_field_enum);
FEL_vector3f_field_ptr make_vector3f_field(FEL_vector3f_field_enum);
```

Request a derived float field or vector field from the manager. The complete list of enums can be found in the FEL User Guide, in FEL_plot3d_field.h, or the PLOT3D User's Manual (as PLOT3D "function numbers", which can be used directly.

```
bool is_mesh() const;
bool is_field() const;
```

Provide run-time type determination for a few key FEL types

```
void set(const FEL_set_keyword_enum, int);
```

Set an option on mesh or any field associated with the manager.

Class: `FEL_plot3d_multi_mesh`

Description:

FEL_plot3d_multi_mesh is a multi-zone mesh where PLOT3D IBLANK data is used to accelerate point location.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_multi_mesh` →
`FEL_plot3d_multi_mesh`.

Public Member Functions:

```
FEL_plot3d_multi_mesh(int, FEL_mesh_ptr*, const char* nm
  = "FEL_plot3d_multi_mesh");
```
Construct a multi-zone mesh.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
bool is_structured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
```

```
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: FEL_plot3d_q

Description:

FEL_plot3d_q represents the standard PLOT3D solution vector.

Public Member Functions:

```
FEL_plot3d_q();
FEL_plot3d_q(float d, FEL_vector3f m, float e);
```

Initialize a plot3d_q object.

```
float operator[](int i) const;
```

Access data members as if a vector.

```
friend FEL_plot3d_q operator-(const FEL_plot3d_q& q);
friend FEL_plot3d_q operator+(const FEL_plot3d_q& lhs,
  const FEL_plot3d_q& rhs);
friend FEL_plot3d_q operator-(const FEL_plot3d_q& lhs,
  const FEL_plot3d_q& rhs);
friend FEL_plot3d_q operator*(float lhs, const
  FEL_plot3d_q& rhs);
friend FEL_plot3d_q operator*(double lhs, const
  FEL_plot3d_q& rhs);
friend FEL_plot3d_q operator*(const FEL_plot3d_q& lhs,
  float rhs);
friend FEL_plot3d_q operator*(const FEL_plot3d_q& lhs,
  double rhs);
```

Do math.

```
friend FEL_plot3d_q operator*(const FEL_matrix33f& lhs,
  const FEL_plot3d_q&;
```

The definition for the product of a 3x3 matrix and a q: ignore the scalar parts of the q, multiply the matrix times the momentum

```
friend bool operator==(const FEL_plot3d_q& lhs, const
  FEL_plot3d_q& rhs);
```

Test for equality.

```
friend ostream& operator<<(ostream&, const
  FEL_plot3d_q&);
```

Write to an ostream.

Class: FEL_pointer<T>

Description:

FEL_pointer is a "smart pointer" that works with classes derived from FEL_reference_counted_object, i.e. objects that are reference counted. The template type T specifies the type of object (particular FEL_reference_counted_object subclass) that a given FEL_pointer instance refers to.

Inheritance Hierarchy:

FEL_pointer_base →
FEL_pointer.

Public Member Functions:

FEL_pointer();
FEL_pointer(T* p);
FEL_pointer(const FEL_pointer<T>& p);

Initialize an FEL_pointer, increment the reference count of p if p != NULL.

~FEL_pointer();

Destruct an FEL_pointer instance, decrement the reference count of the object pointed to if the pointer is non-NULL.

T* operator->() const;

Use an FEL_pointer with the "arrow" syntax, like a C-style pointer.

operator T*() const;

Dereference an FEL_pointer, getting back the raw T* pointer. This operator is used by casting macros which require the internal pointer in order to do a C-style cast. The raw pointer returned by this operator is not accounted for by the reference counting system, use with care!

const FEL_pointer<T>& operator=(T* rhs);
const FEL_pointer<T>& operator=(const FEL_pointer<T>& rhs);

Assign to an FEL_pointer. Increment the reference count of the rhs if non-NULL, decrement the reference count of the object previously referenced by this pointer, if non-NULL.

friend bool operator==(const FEL_pointer<T>& lhs, const FEL_pointer<T>& rhs);
friend bool operator!=(const FEL_pointer<T>& lhs, const FEL_pointer<T>& rhs);
friend bool operator==(const FEL_pointer<T>& lhs, const long rhs);
friend bool operator!=(const FEL_pointer<T>& lhs, const long rhs);

Test for equality or inequality by comparing the internal pointers.

friend ostream& operator<<(ostream& s, const FEL_pointer<T>& p);

Output a hexadecimal representation of the internal pointer.

Typedefs:

FEL_pointer<FEL_core_float_field>
FEL_core_float_field_ptr;

FEL_pointer<FEL_core_plot3d_q_field>
FEL_core_plot3d_q_field_ptr;

FEL_pointer<FEL_core_vector3f_field>
FEL_core_vector3f_field_ptr;

FEL_pointer<FEL_double_field>
FEL_double_field_ptr;

FEL_pointer<FEL_field>
FEL_field_ptr;

FEL_pointer<FEL_fixed_interval_time_series_float_field>
FEL_fixed_interval_time_series_float_field_ptr;

FEL_pointer<FEL_fixed_interval_time_series_vector3f_and_int_field>
FEL_fixed_interval_time_series_vector3f_and_int_field_ptr;

FEL_pointer<FEL_fixed_interval_time_series_vector3f_field>
FEL_fixed_interval_time_series_vector3f_field_ptr;

FEL_pointer<FEL_float_field>
FEL_float_field_ptr;

FEL_pointer<FEL_hexahedral_isoparametric_interpolant>
FEL_hexahedral_isoparametric_interpolant_ptr;

FEL_pointer<FEL_interpolant>
FEL_interpolant_ptr;

FEL_pointer<FEL_matrix33f_field>
FEL_matrix33f_field_ptr;

FEL_pointer<FEL_mesh>
FEL_mesh_ptr;

FEL_pointer<FEL_paged_file>
FEL_paged_file_ptr;

FEL_pointer<FEL_paged_float_field>
FEL_paged_float_field_ptr;

FEL_pointer<FEL_paged_plot3d_density_momentum_field>
FEL_paged_plot3d_density_momentum_field_ptr;

FEL_pointer<FEL_paged_plot3d_q_field>
FEL_paged_plot3d_q_field_ptr;

FEL_pointer<FEL_paged_vector3f_field>
FEL_paged_vector3f_field_ptr;

FEL_pointer<FEL_plot3d_density_momentum_field>

```
    FEL_plot3d_density_momentum_field_ptr;

FEL_pointer<FEL_plot3d_field>
    FEL_plot3d_field_ptr;

FEL_pointer<FEL_plot3d_q_field>
    FEL_plot3d_q_field_ptr;

FEL_pointer<FEL_reference_counted_object>
    FEL_reference_counted_object_ptr;

FEL_pointer<FEL_structured_mesh>
    FEL_structured_mesh_ptr;

FEL_pointer<FEL_time_series_float_field>
    FEL_time_series_float_field_ptr;

FEL_pointer<FEL_time_series_plot3d_q_field>
    FEL_time_series_plot3d_q_field_ptr;

FEL_pointer<FEL_time_series_vector3f_and_int_field>
    FEL_time_series_vector3f_and_int_field_ptr;

FEL_pointer<FEL_time_series_vector3f_field>
    FEL_time_series_vector3f_field_ptr;

FEL_pointer<FEL_unstructured_mesh>
    FEL_unstructured_mesh_ptr;

FEL_pointer<FEL_vector2f_field>
    FEL_vector2f_field_ptr;

FEL_pointer<FEL_vector3d_field>
    FEL_vector3d_field_ptr;

FEL_pointer<FEL_vector3f_and_int_field>
    FEL_vector3f_and_int_field_ptr;

FEL_pointer<FEL_vector3f_field>
    FEL_vector3f_field_ptr;
```

Class: `FEL_pointer_base`

Description:

The class FEL_pointer_base implements essential functionality for FEL_pointer objects. FEL_pointer_base is a friend of FEL_reference_counted_object and uses its friendship privileges to increment, decrement and delete reference counted objects.

Public Member Functions:

Class: `FEL_prismatic_isoparametric_interpolant`

Description:

Supports isoparametric interpolation on prisms.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_interpolant` →
`FEL_isoparametric_interpolant` →
`FEL_isoparametric_3D_interpolant` →
`FEL_prismatic_isoparametric_interpolant`.

Public Member Functions:

`FEL_prismatic_isoparametric_interpolant(const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&);`

Construct the interpolant with the six vertices of a prism.

`const char* get_name() const;`
`void set_name(const char*);`

Access the name of this object.

`bool ok() const;`

Indicates if interpolant was successfully constructed. Workaround for lack of exceptions on many systems.

`int physical_to_computational_coords(const FEL_vector3d&, double&,double&,double&, double tol=1e-6);`

Converts physical coordinates (input as `FEL_vector3d&`) into local computational coordinates, returned in the `double&`'s. The conversion uses Newton-Raphson iteration, with tolerated error `< tol`. Returns 0 if no convergence.

`bool is_hexahedral_isoparametric() const;`

Indicates whether calling object is an `FEL_hexahedral_isoparaemtric_interpolant`.

`FEL_interpolation_enum get_interpolation() const;`

Interface for retrieving interpolation mode.

`bool is_mesh() const;`
`bool is_field() const;`

Provide run-time type determination for a few key FEL types

Class: FEL_product_field<TO, FROM1, FROM2>

Description:

Derived field returning the product of its component fields.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_field →
FEL_typed_field →
FEL_derived_field →
FEL_map_then_interpolate_derived_field2 →
FEL_product_field.

Public Member Functions:

FEL_product_field(FEL_pointer< FEL_typed_field<FROM1> > f1, FEL_pointer< FEL_typed_field<FROM2> > f2, char* nm = "product_field");

Construct this built-in derived field.

int at_phys_pos(const FEL_phys_pos&, const FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&, FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&, FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&, TO[]);

Retrieve field node values, return 1 if successful.

int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);

Retrieve structure with various solution parameters.

const char* get_name() const;
void set_name(const char*);

Access the name of this object.

FEL_pointer< FEL_typed_field<TO> > get_eager_field(const FEL_time& = FEL_time());

Construct a new core field where each node has been eagerly evaluated.

int get_min_max(TO*, TO*, const FEL_time& = FEL_time());

Get the minimum and maximum values of the field (undefined for non-scalar fields).

bool is_core_field() const;
bool is_float_field() const;

bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f [ ]) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

Typedefs:

`FEL_product_field<float, float, float>`
`FEL_product_of_float_field;`

Class: FEL_pyramidal_isoparametric_interpolant

Description:

Supports isoparametric interpolation on pyramids.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_interpolant →
FEL_isoparametric_interpolant →
FEL_isoparametric_3D_interpolant →
FEL_pyramidal_isoparametric_interpolant.

Public Member Functions:

FEL_pyramidal_isoparametric_interpolant(const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&);

Construct the interpolant with the five vertices of a pyramid.

const char* get_name() const;
void set_name(const char*);

Access the name of this object.

bool ok() const;

Indicates if interpolant was successfully constructed. Workaround for lack of exceptions on many systems.

int physical_to_computational_coords(const FEL_vector3d&, double&,double&,double&, double tol=1e-6);

Converts physical coordinates (input as FEL_vector3d&) into local computational coordinates, returned in the double&'s. The conversion uses Newton-Raphson iteration, with tolerated error < tol. Returns 0 if no convergence.

bool is_hexahedral_isoparametric() const;

Indicates whether calling object is an FEL_hexahedral_isoparaemtric_interpolant.

FEL_interpolation_enum get_interpolation() const;

Interface for retrieving interpolation mode.

bool is_mesh() const;
bool is_field() const;

Provide run-time type determination for a few key FEL types

Class: `FEL_q_plot3d_field`

Description:

Field ”manager” based on a q field. This class is designed for unusual or non-file-based fields like transformed fields or fields with some moving and some static zones. If you have a simple steady or file-based unsteady field, paged fields will be more efficient with FEL_steady_plot3d_field or FEL_fixed_interval_time_series_plot3d_field. This class can return over fifty predefined derived fields, in response to an `enum`’ed request, and can take care of any necessary file reading and primitive field caching. This is a virtual base class; use either FEL_q_plot3d_field, FEL_steady_plot3d_field, or FEL_fixed_interval_time_series_plot3d_field to create an instance.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_plot3d_field` →
`FEL_q_plot3d_field`.

Public Member Functions:

```
FEL_q_plot3d_field(FEL_plot3d_q_field_ptr);
```
creates a field ”manager” for the q field.

```
FEL_mesh_ptr get_mesh();
FEL_plot3d_q_field_ptr get_q_field();
FEL_plot3d_density_momentum_field_ptr
  get_density_momentum_field();
FEL_vector3f_field_ptr get_momentum_field();
FEL_float_field_ptr get_density_field();
FEL_float_field_ptr get_energy_field();
```
Retrieve mesh, or PLOT3D primitive fields.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_float_field_ptr make_float_field(FEL_float_field_enum);
FEL_vector3f_field_ptr make_vector3f_field(FEL_vector3f_field_enum);
```
Request a derived float field or vector field from the manager. The complete list of enums can be found in the FEL User Guide, in `FEL_plot3d_field.h`, or the PLOT3D User’s Manual (as PLOT3D ”function numbers”, which can be used directly.

```
bool is_mesh() const;
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
void set(const FEL_set_keyword_enum, int);
```
Set an option on mesh or any field associated with the manager.

Class: `FEL_quotient_field<TO, FROM1, FROM2>`

Description:

Derived field returning the quotient of its component fields.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_derived_field` →
`FEL_map_then_interpolate_derived_field2` →
`FEL_quotient_field`.

Public Member Functions:

```
FEL_quotient_field(FEL_pointer< FEL_typed_field<FROM1> >
  f1, FEL_pointer< FEL_typed_field<FROM2> > f2, char* nm
  = "quotient_field");
```
Construct this built-in derived field.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```
Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```
Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```
Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```
Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
```

bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f [ ]) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`

Get number of timesteps associated with this field.

Class: `FEL_reference_counted_object`

Description:

Objects that are reference counted inherit from this class.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
friend ostream& operator<<(ostream&, const
  FEL_reference_counted_object&);
```

Output to an ostream, using `get_name()` name.

```
bool is_mesh() const;
bool is_field() const;
```

Provide run-time type determination for a few key FEL types

Class: `FEL_regular_axis`

Description:

Inheritance Hierarchy:

`FEL_axis` →
`FEL_regular_axis`.

Public Member Functions:

Class: FEL_regular_mesh

Description:

FEL_regular_mesh represents regular, hexahedral meshes.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_mesh →
FEL_single_mesh →
FEL_structured_mesh →
FEL_axis_aligned_mesh →
FEL_regular_mesh.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
int get_n_zones() const;
```
Return 1 for any subclass of FEL_single_mesh.

```
FEL_mesh_ptr get_zone(int) const;
```
Return the mesh itself the argument is 0, and NULL otherwise.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
int jacobian_at_vertex_cell(const FEL_vertex_cell&,
  FEL_matrix33f*) const;
```
Request Jacobian matrix at given vertex.

```
int contravariant_phys_to_comp_vector(const
  FEL_vertex_cell&, const FEL_vector3f&, FEL_vector3f*)
  const;
int contravariant_phys_to_comp_vector(const
  FEL_structured_pos&, const FEL_vector3f&,
  FEL_vector3f*) const;
```

Convert a contravariant vector in physical space to its equivalent in computational space.

```
bool is_structured_mesh() const;
```

Return true.

```
void set(const FEL_set_keyword_enum, int);
```

Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```

Get a mesh parameter.

```
int card(int k) const;
```

Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```

Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```

Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```

Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```

Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```

Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```

Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```

Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
```

```
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```
Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```
Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```
Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```
Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```
Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```
Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

Class: `FEL_regular_mesh2`

Description:

FEL_regular_mesh2 represents regular, quadrilateral meshes in the Z = 0 plane.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_single_mesh` →
`FEL_structured_mesh` →
`FEL_axis_aligned_mesh` →
`FEL_regular_mesh2`.

Public Member Functions:

`FEL_regular_mesh2(int d0, int d1, float spacing0, float spacing1, char* nm = "regular_mesh2");`

Construct a regular surface mesh with dimensions d0 and d1 and adjacent vertex spacing s0 and s1.

`const char* get_name() const;`
`void set_name(const char*);`

Access the name of this object.

`int get_n_zones() const;`

Return 1 for any subclass of `FEL_single_mesh`.

`FEL_mesh_ptr get_zone(int) const;`

Return the mesh itself the argument is 0, and NULL otherwise.

`bool inside_bounding_box(const FEL_phys_pos& p);`

Return true if p is inside bounding box of mesh.

`int locate_close_vertex_cell(const FEL_phys_pos&, FEL_vertex_cell*) const;`

Locate a vertex close to the given point.

`bool cell_has_collapsed_edge(const FEL_cell&) const;`
`int volume_of_cell(const FEL_cell&, double*) const;`
`int centroid_of_cell(const FEL_cell&, FEL_vector3f*) const;`
`int longest_edge_length_of_cell(const FEL_cell&, float*) const;`
`int closest_vertex_of_cell(const FEL_phys_pos&, const FEL_cell&, FEL_vertex_cell*) const;`

Get geometric properties of a cell.

`bool is_multi_mesh() const;`
`bool is_curvilinear_surface_mesh() const;`
`bool is_mesh() const;`

Provide run-time type determination for a few mesh subclasses.

`bool is_field() const;`

Provide run-time type determination for a few key FEL types

```
int jacobian_at_vertex_cell(const FEL_vertex_cell&,
  FEL_matrix33f*) const;
```
Request Jacobian matrix at given vertex.

```
int contravariant_phys_to_comp_vector(const
  FEL_vertex_cell&, const FEL_vector3f&, FEL_vector3f*)
  const;
int contravariant_phys_to_comp_vector(const
  FEL_structured_pos&, const FEL_vector3f&,
  FEL_vector3f*) const;
```
Convert a contravariant vector in physical space to its equivalent in computational space.

```
bool is_structured_mesh() const;
```
Return true.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
```

```
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```
Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```
Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```
Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```
Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```
Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```
Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

Class: `FEL_regular_xy_irregular_z_mesh`

Description:

FEL_regular_xy_irregular_z_mesh represents meshes that are regular in the X and Y dimensions but irregular in Z.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_single_mesh` →
`FEL_structured_mesh` →
`FEL_axis_aligned_mesh` →
`FEL_regular_xy_irregular_z_mesh`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
int get_n_zones() const;
```
Return 1 for any subclass of `FEL_single_mesh`.

```
FEL_mesh_ptr get_zone(int) const;
```
Return the mesh itself the argument is 0, and NULL otherwise.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
int jacobian_at_vertex_cell(const FEL_vertex_cell&,
  FEL_matrix33f*) const;
```

Request Jacobian matrix at given vertex.

```
int contravariant_phys_to_comp_vector(const
  FEL_vertex_cell&, const FEL_vector3f&, FEL_vector3f*)
  const;
int contravariant_phys_to_comp_vector(const
  FEL_structured_pos&, const FEL_vector3f&,
  FEL_vector3f*) const;
```

Convert a contravariant vector in physical space to its equivalent in computational space.

```
bool is_structured_mesh() const;
```

Return true.

```
void set(const FEL_set_keyword_enum, int);
```

Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```

Get a mesh parameter.

```
int card(int k) const;
```

Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```

Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```

Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```

Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```

Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```

Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```

Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```

Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
```

```
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```

Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: `FEL_remote_field<T>`

Description:

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_remote_field`.

Public Member Functions:

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```

Get the number of cells of specified dimension d.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f
  []) const;
```

Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_time_steps() const;
Get number of timesteps associated with this field.

FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());
Construct a new core field where each node has been eagerly evaluated.

int get_min_max(T*, T*, const FEL_time& = FEL_time());
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Typedefs:

FEL_remote_field<float>
`FEL_remote_float_field;`

FEL_remote_field<FEL_plot3d_q>

```
    FEL_remote_plot3d_q_field;

FEL_remote_field<FEL_vector3f>
    FEL_remote_vector3f_field;
```

Class: `FEL_remote_mesh`

Description:

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_remote_mesh`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
bool is_structured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
```

int coordinates_and_iblank_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f_and_int*) const;
Access coordinates and IBLANK.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Convert time representation.

void begin(FEL_vertex_cell_iter*) const;
Initialize a `vertex_cell` iterator to loop over all the vertices.

void begin(FEL_cell_iter*) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_cell_iter*, int, ...) const;
int begin(FEL_vertex_cell_iter*, int, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter*) const;
Initialize an iterator to test if at end.

bool is_unstructured_mesh() const;
Provide run-time type determination for a few mesh subclasses.

Class: `FEL_rotated_mesh`

Description:

FEL_rotated_mesh classes emulate rotating original coordinates c to new value c' using a matrix m via the transformation c' = m * c.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_transformed_mesh` →
`FEL_rotated_mesh`.

Public Member Functions:

```
FEL_rotated_mesh(FEL_mesh_ptr m, const FEL_matrix33f& r,
  char* = "rotated_mesh");
```
Construct a rotated mesh with mesh m and rotation matrix r.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_mesh_ptr get_zone(int) const;
```
Return a pointer to the zone specified by the argument, with the same rotation applied as by this mesh.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
bool is_structured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
```

```
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: FEL_scattered_vertex_mesh

Description:

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_mesh →
FEL_single_mesh →
FEL_scattered_vertex_mesh.

Public Member Functions:

const char* get_name() const;
void set_name(const char*);
Access the name of this object.

int get_n_zones() const;
Return 1 for any subclass of FEL_single_mesh.

FEL_mesh_ptr get_zone(int) const;
Return the mesh itself the argument is 0, and NULL otherwise.

bool inside_bounding_box(const FEL_phys_pos& p);
Return true if p is inside bounding box of mesh.

int locate_close_vertex_cell(const FEL_phys_pos&, FEL_vertex_cell*) const;
Locate a vertex close to the given point.

bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*) const;
int longest_edge_length_of_cell(const FEL_cell&, float*) const;
int closest_vertex_of_cell(const FEL_phys_pos&, const FEL_cell&, FEL_vertex_cell*) const;
Get geometric properties of a cell.

bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
Provide run-time type determination for a few mesh subclasses.

bool is_field() const;
Provide run-time type determination for a few key FEL types

bool is_structured_mesh() const;
Provide run-time type determination for a few mesh subclasses.

void set(const FEL_set_keyword_enum, int);
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
```

```
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: `FEL_single_mesh`

Description:

FEL_single_mesh represents a mesh that has only one zone. See FEL_multi_mesh.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_single_mesh`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
int get_n_zones() const;
```
Return 1 for any subclass of `FEL_single_mesh`.

```
FEL_mesh_ptr get_zone(int) const;
```
Return the mesh itself the argument is 0, and NULL otherwise.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
bool is_structured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
```

```
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: `FEL_socket`
Description:

Public Member Functions:

Class: `FEL_solution_globals`

Description:

The class FEL_solution_globals contains parameters such as free stream Mach number that are global to a field.

Public Member Functions:

`FEL_solution_globals();`

Initialize.

`bool valid(unsigned flag);`

Test whether a particular parameter is valid.

```
void set_free_stream_mach(float f);
void set_alpha(float f);
void set_reynolds_number(float f);
void set_time(float f);
```

Set one of the parameters.

Class: `FEL_steady_plot3d_field`

Description:
Field manager for a steady field.

Inheritance Hierarchy:
`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_plot3d_field` →
`FEL_steady_plot3d_field`.

Public Member Functions:

```
FEL_steady_plot3d_field(char*, char*, unsigned = 0);
FEL_steady_plot3d_field(FEL_mesh_ptr, char*, unsigned =
  0);
```
creates a field ”manager” for a steady field.

```
FEL_mesh_ptr get_mesh();
FEL_plot3d_q_field_ptr get_q_field();
FEL_plot3d_density_momentum_field_ptr
  get_density_momentum_field();
FEL_vector3f_field_ptr get_momentum_field();
FEL_float_field_ptr get_density_field();
FEL_float_field_ptr get_energy_field();
```
Retrieve mesh, or PLOT3D primitive fields.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_float_field_ptr make_float_field(FEL_float_field_enum);
FEL_vector3f_field_ptr make_vector3f_field(FEL_vector3f_field_enum);
```
Request a derived float field or vector field from the manager. The complete list of enums can be found in the FEL User Guide, in `FEL_plot3d_field.h`, or the PLOT3D User’s Manual (as PLOT3D ”function numbers”, which can be used directly.

```
bool is_mesh() const;
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
void set(const FEL_set_keyword_enum, int);
```
Set an option on mesh or any field associated with the manager.

Class: `FEL_structured_mesh`

Description:

FEL_structured_mesh is the parent class to all structured hexahedral meshes structured quadrilateral surfaces.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_single_mesh` →
`FEL_structured_mesh`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
int get_n_zones() const;
```
Return 1 for any subclass of `FEL_single_mesh`.

```
FEL_mesh_ptr get_zone(int) const;
```
Return the mesh itself the argument is 0, and NULL otherwise.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
int jacobian_at_vertex_cell(const FEL_vertex_cell&,
  FEL_matrix33f*) const;
```
Request Jacobian matrix at given vertex.

```
int contravariant_phys_to_comp_vector(const
  FEL_vertex_cell&, const FEL_vector3f&, FEL_vector3f*)
  const;
int contravariant_phys_to_comp_vector(const
  FEL_structured_pos&, const FEL_vector3f&,
  FEL_vector3f*) const;
```

Convert a contravariant vector in physical space to its equivalent in computational space.

```
bool is_structured_mesh() const;
```

Return true.

```
void set(const FEL_set_keyword_enum, int);
```

Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```

Get a mesh parameter.

```
int card(int k) const;
```

Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```

Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```

Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```

Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```

Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```

Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```

Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```

Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
```

```
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```
Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```
Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```
Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```
Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```
Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```
Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

Class: FEL_structured_pos

Description:

The class FEL_structured_pos represents a point in computational space, using floating-point coordinates. FEL_structured_pos also contains an integer zone and and an FEL_time data member.

Inheritance Hierarchy:

FEL_vector3f → FEL_structured_pos.

Public Member Functions:

```
FEL_time get_time() const;
float get_physical_time();
float get_computational_time();
int get_time_step();
void set_time(const FEL_time& t);
void set_physical_time(float t);
void set_computational_time(float t);
void set_time_step(int t);
void set_time_undefined();
```

Access the time representation.

```
friend ostream& operator<<(ostream&, const
  FEL_structured_pos&);
```

Output an FEL_structured_pos to an ostream.

```
int get_zone() const;
void set_zone(int z);
```

Access the zone data member.

```
friend bool operator==(const FEL_structured_pos& lhs,
  const FEL_structured_pos& rhs);
friend bool operator!=(const FEL_structured_pos& lhs,
  const FEL_structured_pos& rhs);
```

Test for equality, comparing coordinates, zone and time.

Class: `FEL_sum_field<TO, FROM1, FROM2>`

Description:

Derived field returning the sum of its component fields.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_derived_field` →
`FEL_map_then_interpolate_derived_field2` →
`FEL_sum_field`.

Public Member Functions:

```
FEL_sum_field(FEL_pointer< FEL_typed_field<FROM1> > f1,
  FEL_pointer< FEL_typed_field<FROM2> > f2, char* nm =
  "sum_field");
```

Construct this built-in derived field.

```
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, TO*);
int at_cell(const FEL_cell&, TO[]);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, TO*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, TO*);
int at_vertex_cell(const FEL_vertex_cell&, TO*);
int at_structured_pos(const FEL_structured_pos&, TO*);
int at_phys_pos(const FEL_phys_pos&, TO*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  TO[]);
```

Retrieve field node values, return 1 if successful.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Retrieve structure with various solution parameters.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
FEL_pointer< FEL_typed_field<TO> > get_eager_field(const
  FEL_time& = FEL_time());
```

Construct a new core field where each node has been eagerly evaluated.

```
int get_min_max(TO*, TO*, const FEL_time& = FEL_time());
```

Get the minimum and maximum values of the field (undefined for non-scalar fields).

```
bool is_core_field() const;
bool is_float_field() const;
```

bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
Return true if calling object is an instance of corresponding class.

bool varies_with_time() const;
Return true if a time-varying field is in the lineage of calling object.

FEL_mesh_ptr get_mesh() const;
Retrieve a pointer to the mesh associated with this field.

int card(int d) const;
Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f [ ]) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_time_steps() const;
Get number of timesteps associated with this field.

Typedefs:

FEL_sum_field<float, float, float>
FEL_sum_of_float_field;

FEL_sum_field<FEL_vector3f, FEL_vector3f, FEL_vector3f>
FEL_sum_of_vector3f_field;

Class: FEL_tetrahedral_isoparametric_interpolant

Description:

Supports isoparametric interpolation on tetrahedra.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_interpolant →
FEL_isoparametric_interpolant →
FEL_isoparametric_3D_interpolant →
FEL_tetrahedral_isoparametric_interpolant.

Public Member Functions:

FEL_tetrahedral_isoparametric_interpolant(const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&);

Construct the interpolant with the four vertices of a tetrahedron.

const char* get_name() const;
void set_name(const char*);

Access the name of this object.

bool ok() const;

Indicates if interpolant was successfully constructed. Workaround for lack of exceptions on many systems.

int physical_to_computational_coords(const FEL_vector3d&, double&,double&,double&, double tol=1e-6);

Converts physical coordinates (input as FEL_vector3d&) into local computational coordinates, returned in the double&'s. The conversion uses Newton-Raphson iteration, with tolerated error < tol. Returns 0 if no convergence.

bool is_hexahedral_isoparametric() const;

Indicates whether calling object is an FEL_hexahedral_isoparaemtric_interpolant.

FEL_interpolation_enum get_interpolation() const;

Interface for retrieving interpolation mode.

bool is_mesh() const;
bool is_field() const;

Provide run-time type determination for a few key FEL types

Class: FEL_tetrahedral_mesh

Description:

FEL_tetrahedral_mesh is a subclass of FEL_unstructured mesh specific to tetrahedral meshes.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_mesh →
FEL_single_mesh →
FEL_unstructured_mesh →
FEL_tetrahedral_mesh.

Public Member Functions:

FEL_tetrahedral_mesh(int n0, FEL_vector3f* xyz, int n_special_triangles, FEL_vector3i* triangles, int* triangle_ids, int n3, FEL_vector4i* tetrahedra, char* = "tetrahedral_mesh");

Construct a tetrahedral mesh given n0 vertices with coordinates given by xyz. Also accept n_special_triangles with vertices in the array triangles and id's in triangle_ids. Also accept n3 tetrahedra, each tetrahedron defined by 4 integer vertex ids.

const char* get_name() const;
void set_name(const char*);

Access the name of this object.

int get_n_zones() const;

Return 1 for any subclass of FEL_single_mesh.

FEL_mesh_ptr get_zone(int) const;

Return the mesh itself the argument is 0, and NULL otherwise.

bool inside_bounding_box(const FEL_phys_pos& p);

Return true if p is inside bounding box of mesh.

int locate_close_vertex_cell(const FEL_phys_pos&, FEL_vertex_cell*) const;

Locate a vertex close to the given point.

bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*) const;
int longest_edge_length_of_cell(const FEL_cell&, float*) const;
int closest_vertex_of_cell(const FEL_phys_pos&, const FEL_cell&, FEL_vertex_cell*) const;

Get geometric properties of a cell.

bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;

bool is_mesh() const;
Provide run-time type determination for a few mesh subclasses.

bool is_field() const;
Provide run-time type determination for a few key FEL types

bool is_structured_mesh() const;
Provide run-time type determination for a few mesh subclasses.

void set(const FEL_set_keyword_enum, int);
Set a mesh parameter.

int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Get a mesh parameter.

int card(int k) const;
Get the cardinality of k-cells.

int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const FEL_time& = FEL_time());
Get the bounding box of the mesh.

bool on_mesh(const FEL_vertex_cell& v) const;
Return true if v is valid vertex on the mesh.

int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
Get the flat file indices of cell vertices.

int adjacent_cells(const FEL_cell&, int*, FEL_cell[]) const;
Return the cells adjacent to the first argument.

int decomposition_cells(const FEL_cell&, int*, FEL_cell[]) const;
Get the simplicial decomposition cells.

int up_cells(const FEL_cell&, int, int, int*, FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[]) const;
Return incident cells.

int int_to_cell(int i, int d, FEL_cell* c, int s = -1) const;
int cell_to_int(const FEL_cell& c, int* i) const;
Convert between a cell `c` and its canonical enumeration `i`.

int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell* c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&, FEL_cell_interpolant*) const;

```
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```
Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```
Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```
Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```
Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```
Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```
Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```
Return true.

```
int get_n_triangle_sets() const;
void get_triangle_set_ids(int[]) const;
```
Get information about triangle sets specified at mesh construction.

Class: `FEL_tetrahedral_physical_interpolant`

Description:

Support for physical space interpolation on tetrahedra.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_interpolant` →
`FEL_physical_interpolant` →
`FEL_tetrahedral_physical_interpolant`.

Public Member Functions:

`FEL_tetrahedral_physical_interpolant(const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&, const FEL_vector3f&);`

Construct the interpolantr with the four vertices of a tetrahedron.

`const char* get_name() const;`
`void set_name(const char*);`

Access the name of this object.

`bool ok() const;`

Indicates if interpolant was successfully constructed. Workaround for lack of exceptions on many systems.

`bool is_hexahedral_isoparametric() const;`

Indicates whether calling object is an `FEL_hexahedral_isoparaemtric_interpolant.`

`FEL_interpolation_enum get_interpolation() const;`

Interface for retrieving interpolation mode.

`bool is_mesh() const;`
`bool is_field() const;`

Provide run-time type determination for a few key FEL types

Class: FEL_time

Description:

FEL_time can store a time value in one of 3 representations: physical, computational or integer computational (i.e. an integer time step).

Public Member Functions:

FEL_time();

Initialize representation to FEL_TIME_REPRESENTATION_UNDEFINED.

friend ostream& operator<<(ostream& strm, const FEL_time& time);

Write to an ostream.

void set_undefined();

Set the representation to FEL_TIME_REPRESENTATION_UNDEFINED.

void set_physical(float t);
void set_computational(float t);
void set_step(int t);

Set time and representation.

bool defined() const;

Test whether the time is defined.

FEL_time_representation_enum get_representation() const;

Query the object about its current time representation.

float get_physical() const;
float get_computational() const;
int get_step() const;

Get the time in the requested representation.

friend bool operator==(const FEL_time& lhs, const FEL_time& rhs);
friend bool operator!=(const FEL_time& lhs, const FEL_time& rhs);

Test for equality, or lack there of.

Class: `FEL_time_series_field<T>`

Description:

FEL_time_series_field represents time-varying fields via a series of time steps.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_time_varying_field` →
`FEL_time_series_field`.

Public Member Functions:

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```

Get the number of cells of specified dimension d.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f
  []) const;
```

Get physical-space coordinates of specified cell.

`void begin(FEL_vertex_cell_iter* vi) const;`
Initialize a `vertex_cell` iterator to loop over all vertices.

`void begin(FEL_cell_iter* ci) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_vertex_cell_iter* vi, int k, ...)  const;`
`int begin(FEL_cell_iter* ci, int k, ...)  const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter* ci) const;`
Initialize an iterator to test if at end.

`bool is_mesh() const;`
Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int =`
`FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&,`
`FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum,`
`FEL_time*) const;`
Change from one time representation to another.

`FEL_pointer< FEL_typed_field<T> > get_field(int t);`
Get the field associated with a time step t.

`bool load(int);`
`bool load_all();`
`void unload(int);`
`void unload_all();`
`void unload_least_recently_used();`
`void set_working_set_size(int);`
`void set_verbose(bool b);`
`void show_working_set(ostream&);`
Manage the working set directly.

`bool is_time_series_field() const;`
Return true.

`int get_n_time_steps() const;`

Return the number of time steps in the time series.

FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());
Construct a new core field where each node has been eagerly evaluated.

int get_min_max(T*, T*, const FEL_time& = FEL_time());
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Typedefs:

FEL_time_series_field<float>
FEL_time_series_float_field;

FEL_time_series_field<FEL_plot3d_q>
FEL_time_series_plot3d_q_field;

FEL_time_series_field<FEL_vector3f_and_int>
FEL_time_series_vector3f_and_int_field;

FEL_time_series_field<FEL_vector3f>
FEL_time_series_vector3f_field;

Class: `FEL_time_series_mesh`

Description:

The class FEL_time_series_mesh provides support for meshes that vary with time, also known as unsteady meshes.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_time_series_mesh`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_mesh_ptr get_zone(int) const;
```
Return a pointer to the zone specified by the argument.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
bool is_structured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
```

int combined_iblank_at_cell(const FEL_cell&, int*) const;
int coordinates_and_iblank_at_cell(const FEL_cell&, FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f_and_int*) const;
Access coordinates and IBLANK.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Convert time representation.

void begin(FEL_vertex_cell_iter*) const;
Initialize a `vertex_cell` iterator to loop over all the vertices.

void begin(FEL_cell_iter*) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_cell_iter*, int, ...) const;
int begin(FEL_vertex_cell_iter*, int, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter*) const;
Initialize an iterator to test if at end.

bool is_unstructured_mesh() const;
Provide run-time type determination for a few mesh subclasses.

Class: `FEL_time_varying_field<T>`

Description:

FEL_time_varying_field is the parent class to FEL_time_series_field.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_time_varying_field`.

Public Member Functions:

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```

Get the number of cells of specified dimension d.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f
  []) const;
```

Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_time_steps() const;
Get number of timesteps associated with this field.

FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());
Construct a new core field where each node has been eagerly evaluated.

int get_min_max(T*, T*, const FEL_time& = FEL_time());
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Class: `FEL_timer`

Description:

FEL_timer is a high-resolution timer for SGI systems. A much lower-resolution timer is provided as a fallback.

Public Member Functions:

`FEL_timer();`

Initialize a timer.

`void reset();`
`void start();`
`void stop();`

Reset, start and stop the timer.

`double elapsed();`

Return the time elapsed, in microseconds. (Does not reset the timer.)

Class: `FEL_touch_counted_field<T>`

Description:

This field type simply sits on top of another field and records usage statistics from that field. Such statistics can be very informative during application development or tuning, as they may guide decisions about deployment of lazy evaluation, eager evaluation, or paged fields.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_field` →
`FEL_typed_field` →
`FEL_touch_counted_field`.

Public Member Functions:

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```

Get the number of cells of specified dimension d.

int coordinates_at_cell(const FEL_cell&, FEL_vector3f []) const;
Get physical-space coordinates of specified cell.

void begin(FEL_vertex_cell_iter* vi) const;
Initialize a `vertex_cell` iterator to loop over all vertices.

void begin(FEL_cell_iter* ci) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_vertex_cell_iter* vi, int k, ...) const;
int begin(FEL_cell_iter* ci, int k, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter* ci) const;
Initialize an iterator to test if at end.

bool is_mesh() const;
Provide run-time type determination for a few key FEL types

bool is_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_zones() const;
Get number of zones in this field's mesh.

void set(const FEL_set_keyword_enum, int);
int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;
Set an option on this field, or its mesh.

int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;
Get physical-space coordinates of specified cell.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Change from one time representation to another.

bool is_time_series_field() const;
Return true if calling object is an instance of corresponding class.

int get_n_time_steps() const;
Get number of timesteps associated with this field.

FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());
Construct a new core field where each node has been eagerly evaluated.

int get_min_max(T*, T*, const FEL_time& = FEL_time());
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Typedefs:

FEL_touch_counted_field<float>
FEL_touch_counted_float_field;

FEL_touch_counted_field<FEL_plot3d_q>
FEL_touch_counted_plot3d_q_field;

FEL_touch_counted_field<FEL_vector3f>
FEL_touch_counted_vector3f_field;

Class: `FEL_transformed_mesh`

Description:

Transformed mesh subclasses emulate applying rigid transformations to meshes.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_transformed_mesh`.

Public Member Functions:

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
bool is_structured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
```

int coordinates_and_iblank_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f_and_int*) const;
Access coordinates and IBLANK.

int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;
Convert time representation.

void begin(FEL_vertex_cell_iter*) const;
Initialize a `vertex_cell` iterator to loop over all the vertices.

void begin(FEL_cell_iter*) const;
Initialize a cell iterator to loop over all the highest-dimension cells.

int begin(FEL_cell_iter*, int, ...) const;
int begin(FEL_vertex_cell_iter*, int, ...) const;
Initialize an iterator with a series of keyword/value pairs.

void end(FEL_cell_iter*) const;
Initialize an iterator to test if at end.

bool is_unstructured_mesh() const;
Provide run-time type determination for a few mesh subclasses.

Class: `FEL_translated_mesh`

Description:

Translated meshes emulate applying a translation to a mesh.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_transformed_mesh` →
`FEL_translated_mesh`.

Public Member Functions:

`FEL_translated_mesh(FEL_mesh_ptr, const FEL_vector3f& t, char* = "translated_mesh");`
Construct a mesh emulating a translation of t.

`const char* get_name() const;`
`void set_name(const char*);`
Access the name of this object.

`FEL_mesh_ptr get_zone(int) const;`
Return a pointer to the zone specified by the argument, with the same translation applied as by this mesh.

`bool inside_bounding_box(const FEL_phys_pos& p);`
Return true if p is inside bounding box of mesh.

`int locate_close_vertex_cell(const FEL_phys_pos&, FEL_vertex_cell*) const;`
Locate a vertex close to the given point.

`bool cell_has_collapsed_edge(const FEL_cell&) const;`
`int volume_of_cell(const FEL_cell&, double*) const;`
`int centroid_of_cell(const FEL_cell&, FEL_vector3f*) const;`
`int longest_edge_length_of_cell(const FEL_cell&, float*) const;`
`int closest_vertex_of_cell(const FEL_phys_pos&, const FEL_cell&, FEL_vertex_cell*) const;`
Get geometric properties of a cell.

`bool is_multi_mesh() const;`
`bool is_curvilinear_surface_mesh() const;`
`bool is_mesh() const;`
Provide run-time type determination for a few mesh subclasses.

`bool is_field() const;`
Provide run-time type determination for a few key FEL types

`bool is_structured_mesh() const;`
Provide run-time type determination for a few mesh subclasses.

`void set(const FEL_set_keyword_enum, int);`

Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
```

```
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: `FEL_triangle_set`

Description:

FEL_triangle_set represents a set of triangles and the set of vertex faces of the triangles.

Public Member Functions:

`FEL_triangle_set();`

Initialize to defaults.

`~FEL_triangle_set();`

Deallocate buffers.

Class: `FEL_triangle_vertices`

Description:

FEL_triangle_vertices is used to represent the 3 integer vertex indices of a triangle. The class is used by FEL_unstructured_mesh.

Inheritance Hierarchy:

`FEL_vector3i` → `FEL_triangle_vertices`.

Public Member Functions:

`FEL_triangle_vertices();`
`FEL_triangle_vertices(const FEL_vector3i& v3i);`
`FEL_triangle_vertices(int v0, int v1, int v2);`

Initialize.

`friend unsigned FEL_hash(const FEL_triangle_vertices& tv);`

Return a cheap hash.

`friend bool operator==(const FEL_triangle_vertices& lhs, const FEL_triangle_vertices& rhs);`

Test for equality. Triangle indices do not have to be ordered the same in the lhs and rhs.

Class: FEL_triangular_isoparametric_interpolant

Description:

Supports isoparametric interpolation on triangles.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_interpolant →
FEL_isoparametric_interpolant →
FEL_isoparametric_2D_interpolant →
FEL_triangular_isoparametric_interpolant.

Public Member Functions:

FEL_triangular_isoparametric_interpolant(const FEL_vector2f&, const FEL_vector2f&, const FEL_vector2f&);

Construct the interpolant with the three vertices of a triangle.

const char* get_name() const;
void set_name(const char*);

Access the name of this object.

bool ok() const;

Indicates if interpolant was successfully constructed. Workaround for lack of exceptions on many systems.

int physical_to_computational_coords(const FEL_vector2d&, double&,double&, double tol=1e-6);

Converts physical coordinates (input as FEL_vector2d&) into local computational coordinates, returned in the double&'s. The conversion uses Newton-Raphson iteration, with tolerated error < tol. Returns 0 if no convergence.

bool is_hexahedral_isoparametric() const;

Indicates whether calling object is an FEL_hexahedral_isoparaemtric_interpolant.

FEL_interpolation_enum get_interpolation() const;

Interface for retrieving interpolation mode.

bool is_mesh() const;
bool is_field() const;

Provide run-time type determination for a few key FEL types

Class: FEL_typed_fieldT>

Description:

Pure virtual base class for all templated fields. This class declares the type-dependent interface common to all fields – primarily at_*() calls, which provide lazy evaluation of field values at given locations in space and time.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_field →
FEL_typed_field.

Public Member Functions:

```
int at_cell(const FEL_cell&, T[]);
int at_vertex_cell(const FEL_vertex_cell&, T*);
int at_cell_interpolant(const FEL_cell_interpolant&,
  T[]);
int at_structured_pos(const FEL_structured_pos&, T*);
int at_phys_pos(const FEL_phys_pos&, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*, T*);
int at_phys_pos(const FEL_phys_pos&, const
  FEL_cell_interpolant&, T*);
```

Retrieve field node values, return 1 if successful.

```
const char* get_name() const;
void set_name(const char*);
```

Access the name of this object.

```
bool is_core_field() const;
bool is_float_field() const;
bool is_vector3f_field() const;
bool is_plot3d_q_field() const;
```

Return true if calling object is an instance of corresponding class.

```
int get_solution_globals(FEL_solution_globals*);
void set_solution_globals(const FEL_solution_globals&);
```

Access global ”metadata” associated with this field.

```
bool varies_with_time() const;
```

Return true if a time-varying field is in the lineage of calling object.

```
FEL_mesh_ptr get_mesh() const;
```

Retrieve a pointer to the mesh associated with this field.

```
int card(int d) const;
```

Get the number of cells of specified dimension d.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f
  []) const;
```

Get physical-space coordinates of specified cell.

`void begin(FEL_vertex_cell_iter* vi) const;`
Initialize a `vertex_cell` iterator to loop over all vertices.

`void begin(FEL_cell_iter* ci) const;`
Initialize a cell iterator to loop over all the highest-dimension cells.

`int begin(FEL_vertex_cell_iter* vi, int k, ...) const;`
`int begin(FEL_cell_iter* ci, int k, ...) const;`
Initialize an iterator with a series of keyword/value pairs.

`void end(FEL_cell_iter* ci) const;`
Initialize an iterator to test if at end.

`bool is_mesh() const;`
Provide run-time type determination for a few key FEL types

`bool is_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_zones() const;`
Get number of zones in this field's mesh.

`void set(const FEL_set_keyword_enum, int);`
`int get(const FEL_get_keyword_enum, int*, int[], int = FEL_ZONE_UNDEFINED) const;`
Set an option on this field, or its mesh.

`int coordinates_at_vertex_cell(const FEL_vertex_cell&, FEL_vector3f*) const;`
Get physical-space coordinates of specified cell.

`int convert_time(const FEL_time&, FEL_time_representation_enum, FEL_time*) const;`
Change from one time representation to another.

`bool is_time_series_field() const;`
Return true if calling object is an instance of corresponding class.

`int get_n_time_steps() const;`
Get number of timesteps associated with this field.

`FEL_pointer< FEL_typed_field<T> > get_eager_field(const FEL_time& = FEL_time());`
Construct a new core field where each node has been eagerly evaluated.

`int get_min_max(T*, T*, const FEL_time& = FEL_time());`
Get the minimum and maximum values of the field (undefined for non-scalar fields).

Typedefs:

`FEL_typed_field<double>`
`FEL_double_field;`

FEL_typed_field<float>
FEL_float_field;

FEL_typed_field<FEL_matrix33f>
FEL_matrix33f_field;

FEL_typed_field<FEL_plot3d_density_momentum>
FEL_plot3d_density_momentum_field;

FEL_typed_field<FEL_plot3d_q>
FEL_plot3d_q_field;

FEL_typed_field<FEL_vector2f>
FEL_vector2f_field;

FEL_typed_field<FEL_vector3d>
FEL_vector3d_field;

FEL_typed_field<FEL_vector3f_and_int>
FEL_vector3f_and_int_field;

FEL_typed_field<FEL_vector3f>
FEL_vector3f_field;

Class: FEL_unstructured_mesh

Description:

The class FEL_unstructured_mesh represents meshes that are not not structured. Currently FEL unstructured meshes consist of tetrahedra, triangles, edges and vertices.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_mesh →
FEL_single_mesh →
FEL_unstructured_mesh.

Public Member Functions:

```
FEL_unstructured_mesh(int, FEL_vector3f*, int,
  FEL_vector4i*, int, FEL_vector3i*, int*, const char* =
  "unstructured_mesh");
```
Construct an unstructured mesh.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
int get_n_zones() const;
```
Return 1 for any subclass of FEL_single_mesh.

```
FEL_mesh_ptr get_zone(int) const;
```
Return the mesh itself the argument is 0, and NULL otherwise.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```

Provide run-time type determination for a few key FEL types

```
bool is_structured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```

Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Return true.

```
int get_n_triangle_sets() const;
void get_triangle_set_ids(int[]) const;
```

Get information about triangle sets specified at mesh construction.

Class: `FEL_vector<int N, T>`

Description:

A vector templated on both the number of components N and their type T.

Public Member Functions:

```
FEL_vector();
FEL_vector(T dat[]);
```
Initialize a vector.

```
friend ostream& operator<<(ostream& strm, const
  FEL_vector<N,T>& v);
```
Write to an ostream.

```
T& operator[](int i);
const T& operator[](int i) const;
```
Access vector components.

```
friend FEL_vector<N,T> operator*(double lhs, const
  FEL_vector<N,T>& rhs);
friend FEL_vector<N,T> operator+(const FEL_vector<N,T>&
  lhs, const FEL_vector<N,T>& rhs);
friend FEL_vector<N,T> operator-(const FEL_vector<N,T>&
  v);
friend FEL_vector<N,T> operator-(const FEL_vector<N,T>&
  lhs, const FEL_vector<N,T>& rhs);
friend FEL_vector<N,T> operator*(const FEL_vector<N,T>&
  lhs, double rhs);
```
Do vector math.

```
friend bool operator==(const FEL_vector<N,T>& lhs,
  const FEL_vector<N,T>& rhs);
```
Test for equality.

Class: FEL_vector2<T>

Description:

A two component vector class.

Public Member Functions:

FEL_vector2();
FEL_vector2(const T& d0, const T& d1);

Initialize a vector.

friend ostream& operator<<(ostream& strm, const FEL_vector2<T>& v);

Write to ostream.

void set(const T& d0, const T& d1);

Access vector data members.

friend bool operator==(const FEL_vector2<T>& lhs, const FEL_vector2<T>&;
friend bool operator!=(const FEL_vector2<T>& lhs, const FEL_vector2<T>&;

Test for equality and inequality.

FEL_vector2<T>& operator*=(const T& f);
FEL_vector2<T>& operator/=(const T& f);
friend FEL_vector2<T> operator+(const FEL_vector2<T>& lhs, const FEL_vector2<T>& rhs);
friend FEL_vector2<T> operator-(const FEL_vector2<T>& lhs, const FEL_vector2<T>& rhs);
friend FEL_vector2<T> operator*(const double lhs, const FEL_vector2<T>& rhs);
friend FEL_vector2<T> operator*(const FEL_vector2<T>& lhs, const double rhs);

Do vector math.

T* v();

Expose the raw representation for calls to libraries expecting T*.

T& operator[](int i);
const T& operator[](int i) const;

Access vector data members.

Typedefs:

FEL_vector2<double>
FEL_vector2d;

FEL_vector2<float>
FEL_vector2f;

FEL_vector2<int>
FEL_vector2i;

Class: FEL_vector3<T>

Description:

A three component vector.

Public Member Functions:

```
FEL_vector3();
FEL_vector3(const T& d0, const T& d1, const T& d2);
```
Initialize a 3 component vector.

```
friend ostream& operator<<(ostream& strm, const
  FEL_vector3<T>& v);
```
Write to an ostream.

```
void set(const T& d0, const T& d1, const T& d2);
```
Access vector members.

```
friend bool operator==(const FEL_vector3<T>& lhs, const
  FEL_vector3<T>&;
friend bool operator!=(const FEL_vector3<T>& lhs, const
  FEL_vector3<T>&;
```
Test for equality and inequality.

```
FEL_vector3<T>& operator+=(const FEL_vector3<T>& v);
FEL_vector3<T>& operator-=(const FEL_vector3<T>& v);
friend FEL_vector3<T> operator-(const FEL_vector3<T>&
  v);
friend FEL_vector3<T> operator+(const FEL_vector3<T>&
  lhs, const FEL_vector3<T>& rhs);
```
Do vector math.

```
T* v();
```
Expose the raw representation for calls to libraries expecting T*.

```
T& operator[](int i);
const T& operator[](int i) const;
```

Typedefs:

FEL_vector3<double>
FEL_vector3d;

FEL_vector3<float>
FEL_vector3f;

FEL_vector3<int>
FEL_vector3i;

Class: `FEL_vector3f_and_int`

Description:

The class FEL_vector3f_and_int is used to represent coordinates and an IBLANK paired together.

Inheritance Hierarchy:

`FEL_vector3f` →
`FEL_vector3f_and_int`.

Public Member Functions:

`FEL_vector3f_and_int();`
`FEL_vector3f_and_int(const float& d0, const float& d1, const float& d2, int ib);`
Initialize an `FEL_vector3f_and_int`.

`friend ostream& operator<<(ostream&, const FEL_vector3f_and_int&);`
Output an `FEL_vector3f_and_int` to an ostream.

`friend FEL_vector3f_and_int operator*(float lhs, const FEL_vector3f_and_int& rhs);`
`friend FEL_vector3f_and_int operator*(double lhs, const FEL_vector3f_and_int& rhs);`
Multiply by a scalar, int member copied from rhs.

`friend FEL_vector3f_and_int operator+(const FEL_vector3f_and_int& lhs, const FEL_vector3f_and_int& rhs);`
Add two items, using `FEL_iblank_combine()` to compute int in result.

Class: `FEL_vector4<T>`

Description:

Public Member Functions:

```
FEL_vector4();
FEL_vector4(const T& d0, const T& d1, const T& d2,
  const T& d3);
```
Initialize a 4 component vector.

```
friend ostream& operator<<(ostream& strm, const
  FEL_vector4<T>& v);
```
Write to an ostream.

```
void set(const T& d0, const T& d1, const T& d2, const
  T& d3);
```
Access vector components.

```
friend bool operator==(const FEL_vector4<T>& lhs, const
  FEL_vector4<T>&;
friend bool operator!=(const FEL_vector4<T>& lhs, const
  FEL_vector4<T>&;
```
Test for equality and inequality.

```
FEL_vector4<T>& operator-=(const FEL_vector4<T>& v);
FEL_vector4<T>& operator+=(const FEL_vector4<T>& v);
FEL_vector4<T>& operator*=(const T& f);
FEL_vector4<T>& operator/=(const T& f);
friend FEL_vector4<T> operator-(const FEL_vector4<T>&
  v);
friend FEL_vector4<T> operator+(const FEL_vector4<T>&
  lhs, const FEL_vector4<T>& rhs);
friend FEL_vector4<T> operator-(const FEL_vector4<T>&
  lhs, const FEL_vector4<T>& rhs);
friend FEL_vector4<T> operator*(float lhs, const
  FEL_vector4<T>& rhs);
friend FEL_vector4<T> operator*(const FEL_vector4<T>&
  lhs, float rhs);
friend FEL_vector4<T> operator*(double lhs, const
  FEL_vector4<T>& rhs);
friend FEL_vector4<T> operator*(const FEL_vector4<T>&
  lhs, double rhs);
```
Do vector math.

```
T* v();
```
Expose the raw representation for calls to libraries expecting T*.

```
T& operator[](int i);
const T& operator[](int i) const;
```
Access vector components.

Typedefs:

FEL_vector4<double>
FEL_vector4d;

FEL_vector4<float>
FEL_vector4f;

FEL_vector4<int>
FEL_vector4i;

Class: FEL_vector8<T>

Description:

An 8 component vector.

Public Member Functions:

```
FEL_vector8();
FEL_vector8(const T& d0, const T& d1, const T& d2,
  const T& d3, const T& d4, const T& d5, const T& d6,
  const T& d7);
```

Initialize a vector.

```
friend ostream& operator<<(ostream& strm, const
  FEL_vector8<T>& v);
```

Write to an ostream.

```
void set(const T& d0, const T& d1, const T& d2, const
  T& d3, const T& d4, const T& d5, const T& d6, const
  T& d7);
```

Access vector components.

```
T* v();
```

Expose the raw representation for calls to libraries expecting T*.

```
friend bool operator==(const FEL_vector8<T>& lhs, const
  FEL_vector8<T>&;
friend bool operator!=(const FEL_vector8<T>& lhs, const
  FEL_vector8<T>&;
```

Test for equality and inequality.

```
FEL_vector8<T>& operator-=(const FEL_vector8<T>& v);
FEL_vector8<T>& operator+=(const FEL_vector8<T>& v);
FEL_vector8<T>& operator*=(const T& f);
FEL_vector8<T>& operator/=(const T& f);
friend FEL_vector8<T> operator-(const FEL_vector8<T>&
  v);
friend FEL_vector8<T> operator+(const FEL_vector8<T>&
  lhs, const FEL_vector8<T>& rhs);
friend FEL_vector8<T> operator-(const FEL_vector8<T>&
  lhs, const FEL_vector8<T>& rhs);
friend FEL_vector8<T> operator*(float lhs, const
  FEL_vector8<T>& rhs);
friend FEL_vector8<T> operator*(const FEL_vector8<T>&
  lhs, float rhs);
friend FEL_vector8<T> operator*(double lhs, const
  FEL_vector8<T>& rhs);
friend FEL_vector8<T> operator*(const FEL_vector8<T>&
  lhs, double rhs);
```

Do vector math.

```
T& operator[](int i);
const T& operator[](int i) const;
```

Access vector components.

Typedefs:

FEL_vector8<double>
FEL_vector8d;

FEL_vector8<float>
FEL_vector8f;

FEL_vector8<int>
FEL_vector8i;

Class: `FEL_vertex_cell`

Description:

The FEL_vertex_cell class represents vertices. FEL_vertex_cell inherits almost all its functionality from FEL_cell.

Inheritance Hierarchy:

`FEL_cell` →
`FEL_vertex_cell`.

Public Member Functions:

```
FEL_vertex_cell();
FEL_vertex_cell(int i, int j, int k);
FEL_vertex_cell(const FEL_vector3i& i, int zn, const
  FEL_time& t);
FEL_vertex_cell(const FEL_structured_pos& sp);
FEL_vertex_cell(const FEL_cell& c);
```

Initialize a vertex cell.

```
void set_type(FEL_cell_type_enum t);
short get_subid() const;
void set_subid(short si);
FEL_vector3i get_ijk() const;
int get_ijk(int i) const;
void set_ijk(const FEL_vector3i& i);
void set_ijk(int i, int j, int k);
int& operator[](int i);
const int& operator[](int i) const;
int get_unstructured_id() const;
void set_unstructured_id(int i);
int get_zone() const;
void set_zone(int z);
```

Access the cell data members.

```
float get_physical_time() const;
float get_computational_time() const;
int get_time_step() const;
FEL_time get_time() const;
void set_time(const FEL_time& t);
void set_physical_time(float t);
void set_computational_time(float t);
void set_time_step(int t);
void set_time_undefined();
```

Access the time representation.

```
int get_dim() const;
```

Get the dimension of the cell.

```
int get_n_nodes() const;
int get_n_vertices() const;
```

Get the number of nodes, or the number of vertices.

```
bool simplicial() const;
```
Return true if the cell is a vertex, edge, triangle or tetrahedron.

```
FEL_vertex_cell& operator+=(const FEL_vector3i& rhs);
FEL_vertex_cell& operator-=(const FEL_vector3i& rhs);
friend FEL_vertex_cell operator+(const FEL_vertex_cell&
  v, const FEL_vector3i& ijk);
friend FEL_vertex_cell operator-(const FEL_vertex_cell&
  v, const FEL_vector3i& ijk);
```
Do math with the ijk data member.

Class: `FEL_vertex_cell_iter`

Description:

Inheritance Hierarchy:

`FEL_cell_iter` $\rightarrow$
`FEL_vertex_cell_iter`.

Public Member Functions:

`bool done() const;`
Test whether the iterator is done.

`const FEL_cell& operator*();`
Dereference the iterator.

`void operator++();`
`void operator++(int);`
Increment the iterator.

`int get_zone();`
`void set_zone(int zn);`
Access the cell zone.

`void set_physical_time(float pt);`
`void set_computational_time(float ct);`
`FEL_time get_time();`
Access the time of the cell.

`int card() const;`
Return the number of cells that this iterator is initialized to produce (not supported if the iterator is initialized to loop over mulitiple zones).

`void set_time(const FEL_time& t);`
`void set_time_step(int t);`
Access the time of the cell.

Class: FEL_x_rotated_mesh

Description:

Emulate rotating a mesh about the x-axis.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_mesh →
FEL_transformed_mesh →
FEL_rotated_mesh →
FEL_x_rotated_mesh.

Public Member Functions:

FEL_x_rotated_mesh(FEL_mesh_ptr m, float a, char* nm = "x_rotated_mesh");

Construct a mesh rotated a degrees about the x axis.

const char* get_name() const;
void set_name(const char*);

Access the name of this object.

FEL_mesh_ptr get_zone(int) const;

Return a pointer to the zone specified by the argument, with the same rotation applied as by this mesh.

bool inside_bounding_box(const FEL_phys_pos& p);

Return true if p is inside bounding box of mesh.

int locate_close_vertex_cell(const FEL_phys_pos&, FEL_vertex_cell*) const;

Locate a vertex close to the given point.

bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*) const;
int longest_edge_length_of_cell(const FEL_cell&, float*) const;
int closest_vertex_of_cell(const FEL_phys_pos&, const FEL_cell&, FEL_vertex_cell*) const;

Get geometric properties of a cell.

bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;

Provide run-time type determination for a few mesh subclasses.

bool is_field() const;

Provide run-time type determination for a few key FEL types

bool is_structured_mesh() const;

Provide run-time type determination for a few mesh subclasses.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
```

```
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: `FEL_y_rotated_mesh`

Description:

Emulate rotating a mesh about the y-axis.

Inheritance Hierarchy:

`FEL_reference_counted_object` →
`FEL_mutex_reference_counted_object` →
`FEL_mesh` →
`FEL_transformed_mesh` →
`FEL_rotated_mesh` →
`FEL_y_rotated_mesh`.

Public Member Functions:

```
FEL_y_rotated_mesh(FEL_mesh_ptr m, float a, char* nm =
  "y_rotated_mesh");
```
Construct a mesh rotated a degrees about the y axis.

```
const char* get_name() const;
void set_name(const char*);
```
Access the name of this object.

```
FEL_mesh_ptr get_zone(int) const;
```
Return a pointer to the zone specified by the argument, with the same rotation applied as by this mesh.

```
bool inside_bounding_box(const FEL_phys_pos& p);
```
Return true if p is inside bounding box of mesh.

```
int locate_close_vertex_cell(const FEL_phys_pos&,
  FEL_vertex_cell*) const;
```
Locate a vertex close to the given point.

```
bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*)
  const;
int longest_edge_length_of_cell(const FEL_cell&, float*)
  const;
int closest_vertex_of_cell(const FEL_phys_pos&, const
  FEL_cell&, FEL_vertex_cell*) const;
```
Get geometric properties of a cell.

```
bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
bool is_field() const;
```
Provide run-time type determination for a few key FEL types

```
bool is_structured_mesh() const;
```
Provide run-time type determination for a few mesh subclasses.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
```

```
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: FEL_z_rotated_mesh

Description:

Emulate rotating a mesh about the z-axis.

Inheritance Hierarchy:

FEL_reference_counted_object →
FEL_mutex_reference_counted_object →
FEL_mesh →
FEL_transformed_mesh →
FEL_rotated_mesh →
FEL_z_rotated_mesh.

Public Member Functions:

FEL_z_rotated_mesh(FEL_mesh_ptr m, float a, char* nm = "z_rotated_mesh");

Construct a mesh rotated a degrees about the z axis.

const char* get_name() const;
void set_name(const char*);

Access the name of this object.

FEL_mesh_ptr get_zone(int) const;

Return a pointer to the zone specified by the argument, with the same rotation applied as by this mesh.

bool inside_bounding_box(const FEL_phys_pos& p);

Return true if p is inside bounding box of mesh.

int locate_close_vertex_cell(const FEL_phys_pos&, FEL_vertex_cell*) const;

Locate a vertex close to the given point.

bool cell_has_collapsed_edge(const FEL_cell&) const;
int volume_of_cell(const FEL_cell&, double*) const;
int centroid_of_cell(const FEL_cell&, FEL_vector3f*) const;
int longest_edge_length_of_cell(const FEL_cell&, float*) const;
int closest_vertex_of_cell(const FEL_phys_pos&, const FEL_cell&, FEL_vertex_cell*) const;

Get geometric properties of a cell.

bool is_multi_mesh() const;
bool is_curvilinear_surface_mesh() const;
bool is_mesh() const;

Provide run-time type determination for a few mesh subclasses.

bool is_field() const;

Provide run-time type determination for a few key FEL types

bool is_structured_mesh() const;

Provide run-time type determination for a few mesh subclasses.

```
void set(const FEL_set_keyword_enum, int);
```
Set a mesh parameter.

```
int get(const FEL_get_keyword_enum, int*, int[], int =
  FEL_ZONE_UNDEFINED) const;
```
Get a mesh parameter.

```
int card(int k) const;
```
Get the cardinality of k-cells.

```
int get_bounding_box(FEL_vector3f*, FEL_vector3f*, const
  FEL_time& = FEL_time());
```
Get the bounding box of the mesh.

```
bool on_mesh(const FEL_vertex_cell& v) const;
```
Return true if v is valid vertex on the mesh.

```
int get_node_index(const FEL_vertex_cell&) const;
void get_node_indices(const FEL_cell&, int[]) const;
```
Get the flat file indices of cell vertices.

```
int adjacent_cells(const FEL_cell&, int*, FEL_cell[])
  const;
```
Return the cells adjacent to the first argument.

```
int decomposition_cells(const FEL_cell&, int*,
  FEL_cell[]) const;
```
Get the simplicial decomposition cells.

```
int up_cells(const FEL_cell&, int, int, int*,
  FEL_cell[], int = -1) const;
int down_cells(const FEL_cell&, int, int*, FEL_cell[])
  const;
```
Return incident cells.

```
int int_to_cell(int i, int d, FEL_cell* c, int s = -1)
  const;
int cell_to_int(const FEL_cell& c, int* i) const;
```
Convert between a cell `c` and its canonical enumeration `i`.

```
int locate(const FEL_phys_pos& p, FEL_cell* c) const;
int locate(const FEL_phys_pos& p, FEL_cell&, FEL_cell*
  c) const;
int set_interpolant(FEL_cell_interpolant*) const;
int set_interpolant(const FEL_cell_interpolant&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant*) const;
int locate_and_set_interpolant(const FEL_phys_pos&,
  FEL_cell_interpolant&, FEL_cell_interpolant*) const;
```
Locate and set interpolant.

```
int coordinates_at_cell(const FEL_cell&, FEL_vector3f[])
  const;
```

```
int coordinates_at_vertex_cell(const FEL_vertex_cell&,
  FEL_vector3f*) const;
int coordinates_at_structured_pos(const
  FEL_structured_pos&, FEL_vector3f*) const;
int iblank_at_cell(const FEL_cell&, int[]) const;
int iblank_at_vertex_cell(const FEL_vertex_cell&, int*)
  const;
int combined_iblank_at_cell(const FEL_cell&, int*)
  const;
int coordinates_and_iblank_at_cell(const FEL_cell&,
  FEL_vector3f_and_int[]) const;
int coordinates_and_iblank_at_vertex_cell(const
  FEL_vertex_cell&, FEL_vector3f_and_int*) const;
```

Access coordinates and IBLANK.

```
int convert_time(const FEL_time&, FEL_time_representation_enum,
  FEL_time*) const;
```

Convert time representation.

```
void begin(FEL_vertex_cell_iter*) const;
```

Initialize a `vertex_cell` iterator to loop over all the vertices.

```
void begin(FEL_cell_iter*) const;
```

Initialize a cell iterator to loop over all the highest-dimension cells.

```
int begin(FEL_cell_iter*, int, ...)  const;
int begin(FEL_vertex_cell_iter*, int, ...)  const;
```

Initialize an iterator with a series of keyword/value pairs.

```
void end(FEL_cell_iter*) const;
```

Initialize an iterator to test if at end.

```
bool is_unstructured_mesh() const;
```

Provide run-time type determination for a few mesh subclasses.

Class: `TSL_base`

Description:

TSL_base is designed to be a repository for methods and data common to all task synchronization classes.

Public Member Functions:

Class: `TSL_mutex`

Description:

TSL_mutex provides a simple, spinning mutual exclusion lock.

Inheritance Hierarchy:

`TSL_base` →
`TSL_mutex`.

Public Member Functions:

`void lock();`

Acquire the lock.

`void unlock();`

Release the lock.