Doc++ for Interpol version 1.04

CRS4
Centro di Ricerca, Sviluppo e Studi Superiori in Sardegna
Sesta Strada, Ovest
Zona Industriale Macchiareddu
09010 Uta (Cagliari) Italy

E-mail: scheinin@crs4.it

Contents

1 TNT 11

1.1 Vector — *Basic TNT numerical vector.* 11

2 CastToSelfType — *Contains a method to cast a base class to a derived class* 12

2.1 cast_to_self_type — *Casts a base class U to a derived class T.* 12

3 invert_matrix — *Inverts a matrix using LAPACK routines.* 13

4 LimitRange — *Converts between various primitive number types.* 14

4.1 methods 14

5 Number — *Can represent any type of number.* 15

5.1 cast underlying rep to Number pointer 15

5.1.1 usual methods 16

5.1.2 virtual methods 17

6 operator+ 19

7 operator- 20

8 operator* 21

9 operator/ 22

10 GetNumber — *template parameterized getNumber()* 23

10.1 getNumber — *Gets a primitive number from wrapper class Number.* 23

11 NumberTyped — *Contains one scalar of type T.* 24

11.1 type definitions 24

11.2 data 29

11.3 usual methods 25

11.4 methods 26

11.5 arithmetic methods 26

11.6 helper methods 27

11.7 virtual methods of LinAlgScalar 27

11.7.1 operator- — *A virtual function of class LinAlgScalar.* 28

11.8 virtual methods of Number 28

12 operator+ 30

13 operator- 31

14 operator* 32

15 operator/ 33

16 operator<< 34

17 operator>> 35

18 LinAlgScalar — *A scalar for linear algebra, independent of numeric type* 36

18.1 data 38

18.1.1 lin_alg_scalar_ — *Is non-zero only when the actual, highest level class is the base class* 38

18.2 usual methods 36

18.2.1 ~LinAlgScalar — *Each instantation has its own instantation of a LinAlgScalar* lin_alg_scalar_ so it is deleted when the object is deleted* 37

18.3 virtual methods 37

18.4 public methods 38

19 operator+ 39

20 **operator-** 40
21 **operator*** 41
22 **operator/** 42
23 **LinAlgScalarTyped — *The class LinAlgScalarTyped<T> contains one scalar of type T*** 43
23.1 type definitions 44
23.2 data 46
23.3 methods 44
23.3.1 operator- — *A virtual function of class LinAlgScalar.* 46

24 **operator+** 47
25 **operator-** 48
26 **operator*** 49
27 **operator/** 50
28 **operator<<** 51
29 **operator>>** 52
30 **LinAlgVector — *Base class for a vector used in linear algebra*** 53
30.1 data 56
30.1.1 lin_alg_vector_ — *Is non-zero only when the actual, highest level class is the base class* 56
30.2 usual methods 53
30.2.1 LinAlgVector — *Each instantation of LinAlgVector has its own lin_alg_vector_, created using clone().* 54
30.3 virtual methods 54
30.4 public methods 55

31 **operator+** 57
32 **operator-** 58
33 **operator*** 59
34 **operator+** 60
35 **operator-** 61
36 **operator*** 62
37 **operator+** 63
38 **operator-** 64
39 **operator*** 65
40 **TNTVect — *A TNT vector with some modifications.*** 66
40.1 copy and assignment helper methods 69
40.1.1 convert — *Copy of data members* 69
40.2 fast copy and set 69
40.3 usual methods 67
40.4 methods of TNT Vector that return **this** 68
40.5 new methods not in TNT Vector 68
40.6 I/O 68

41 **operator+ — *TNTVect sum.*** 70
42 **operator- — *TNTVect difference.*** 71
43 **operator* — *TNTVect component by component product.*** 72
44 **dot_prod — *TNTVect inner product.*** 73
45 **operator<< — *TNTVect write to standard output.*** 74
46 **operator>> — *TNTVect read from standard input.*** 75
47 **ReadOnlyNumArray — *An array that can be declared Read Only*** 76
47.1 data 77
47.2 usual methods 76
47.3 methods 77

48 Timer — *A stopwatch* 78
48.1 data 79
48.2 public typedef and data 78
48.3 usual methods 78
48.4 methods 79

49 operator<< 80
50 operator>> 81
51 operator<< 82
52 operator>> 83
53 operator<< 84
54 operator>> 85
55 Vector1 — *Include access by functions used by Vertex in order to generalize algorithms* ... 86
55.1 public methods 86

56 Vector2 — *Include access by functions used by Vertex in order to generalize algorithms* ... 87
56.1 public methods 87

57 Vector3 — *Include access by functions used by Vertex in order to generalize algorithms* ... 88
57.1 public methods 88

58 LinAlgVectorSpace 90
58.1 casting to actual type 91
58.2 usual methods 90
58.3 virtual methods 90

59 VecSpecificDim — *T can be Vector1, Vector2, or Vector3* 92
59.1 data 96
59.2 usual methods 93
59.2.1 VecSpecificDim — *default constructor* 93
59.3 static methods 93
59.4 access to data 94
59.5 operators 94
59.6 virtual functions of LinAlgVector 94
59.7 virtual functions of LinAlgVectorSpace 95
59.8 linear algebra functions 95
59.9 virtual functions of LinAlgVector 96

60 operator+ 97
61 operator- 98
62 operator* 99
63 operator+ 100
64 operator- 101
65 operator* 102
66 operator+ 103
67 operator- 104
68 operator* 105
69 operator<< <Vector1> 106
70 operator>> <Vector1> 107
71 operator<< <Vector2> 108
72 operator>> <Vector2> 109
73 operator<< <Vector3> 110
74 operator>> <Vector3> 111
75 BsplineEquations — *B splines for interpolation.* 112
75.1 basic equations 112

75.1.1 bsplinepair2 — *input x is centered between two peaks* 114
75.1.2 bsplinepair2_derivative — *input x is centered between two peaks* 114
75.1.3 bsplinepair2_integral — *input x is centered between two peaks* 114
75.1.4 bsplinepair2_integral — *input x is centered between two peaks* 114
75.1.5 bsplinepair3 — *input x is centered between two peaks* 114
75.1.6 bsplinepair3_derivative — *input x is centered between two peaks* 115
75.1.7 bsplinepair3_integral — *input x is centered between two peaks* 115
75.1.8 bsplinepair3_integral — *input x is centered between two peaks* 115
75.1.9 bsplinepair4 — *input x is centered between two peaks* 115
75.1.10 bsplinepair4_derivative — *input x is centered between two peaks* 116
75.1.11 bsplinepair4_integral — *input x is centered between two peaks* 116
75.1.12 bsplinepair4_integral — *input x is centered between two peaks* 116

76 Bspline — *B-spline with specific width.* 117
76.1 basic equations 120
76.2 copy and assignment helper methods 118
76.2.1 convert — *Copy of data members* 119
76.2.2 convert_tree — *Call convert_tree on each parent class then call convert* ... 119
76.3 methods 119
76.4 usual methods 117
76.4.1 Bspline — *Default constructor.* 118
76.5 methods 118

77 StencilHandle — *StencilHandle takes control of a pointer.* 121
77.1 methods 122
77.2 data 122
77.3 methods 121

78 POINT_VALUE_MODE — *Value set in stencil_matrix.C* 123
79 BOX_VALUE_MODE — *Value set in stencil_matrix.C* 124
80 PRECISION_LEVEL1 — *Value set in Field/field_interpol_algorithms.C* 125
81 PRECISION_LEVEL2 — *Value set in Field/field_interpol_algorithms.C* 126
82 PRECISION_LEVEL3 — *Value set in Field/field_interpol_algorithms.C* 127
83 PRECISION_LEVEL4 — *Value set in Field/field_interpol_algorithms.C* 128
84 MAX_STENCIL_SITES 129
85 StencilSites — *A stencil of sites.* 130
85.1 methods 136
85.2 copy and assignment helper methods 133
85.2.1 convert — *Copy of data members* 133
85.2.2 convert_tree — *Call convert_tree on each parent class then call convert* ... 133
85.3 methods 134
85.4 static methods 134
85.5 data 134
85.5.1 initialized — *Whether initialized.* 136
85.6 constructors 131
85.6.1 StencilSites — *default constructor* 131
85.7 usual methods 131
85.8 Data 131
85.8.1 stencil_sites_x — *X positions of the stencil sites* 132
85.8.2 stencil_sites_y — *Y positions of the stencil sites* 132
85.8.3 stencil_sites_z — *Z positions of the stencil sites* 132
85.9 static methods 133

86 ArbitrarySites — *An arbitrary group of sites.* 137
86.1 copy and assignment helper methods 142
86.1.1 convert — *Copy of data members* 143
86.1.2 convert_tree — *Call convert_tree on each parent class then call convert* 143
86.2 data 143
86.3 methods 143
86.4 usual methods 137
86.4.1 ArbitrarySites — *Constructor.* 138
86.4.2 ArbitrarySites — *Constructor.* 139
86.4.3 ArbitrarySites — *Constructor.* 139
86.4.4 ArbitrarySites — *Constructor.* 139
86.4.5 ArbitrarySites — *Constructor.* 139
86.4.6 ArbitrarySites — *Constructor.* 140
86.4.7 ArbitrarySites — *Constructor.* 140
86.4.8 ArbitrarySites — *Constructor.* 140
86.4.9 ArbitrarySites — *Constructor.* 140
86.5 Data 141
86.5.1 stencil_sites_x — *X positions of the sites* 141
86.5.2 stencil_sites_y — *Y positions of the sites* 141
86.5.3 stencil_sites_z — *Z positions of the sites* 141
86.6 methods 142
86.6.1 getSize — *Get total size.* 142
86.6.2 getDimension — *Get number of dimensions.* 142

87 StencilSitesTag — *Tag for stencil sites.* 145
87.1 data 146
87.2 methods. 145
87.2.1 getIntegerValue — *Get tag value as an integer.* 146

88 StencilParams — *Basic parameters for any stencil.* 147
88.1 copy and assignment helper methods 149
88.1.1 convert — *Copy of data members* 150
88.1.2 convert_tree — *Call convert_tree on each parent class then call convert* 150
88.2 data 150
88.3 methods 151
88.3.1 check_stencil_tag — *check that tag is valid.* 151
88.4 constructors. 147
88.4.1 StencilParams — *Default constructor.* 148
88.5 usual methods. 148
88.6 methods. 148
88.6.1 getTag — *Get tag value.* 149
88.6.2 getSize — *Get total size.* 149
88.6.3 getDimension — *Get number of dimensions (2 or 3).* 149

89 TermsTag — *Tag for polynomial terms.* 152
89.1 data. 152
89.2 methods 152
89.2.1 usual methods 153
89.2.2 data access methods 153
89.2.3 static methods 154

90 PrecisionChoice — *Holds choice of stencil and choice of basis for interpolation* 155

90.1 data . . . 155

91 StencilTerms — *Polynomial terms for a given stencil.* . . . 156
91.1 copy and assignment helper methods . . . 161
91.1.1 convert — *Copy of data members* . . . 161
91.1.2 convert_tree — *Call convert_tree on each parent class then call convert* . . . 161
91.2 data . . . 162
91.3 methods . . . 162
91.4 Constructors . . . 156
91.4.1 StencilTerms — *Default constructor.* . . . 157
91.5 Usual methods . . . 157
91.6 data . . . 157
91.6.1 terms — *An array of either the values of the polynomial terms at a location or the values of the polynomial terms with each averaged over a box* . . . 158
91.6.2 xterms — *The x values of an array of vectors for the gradient* . . . 158
91.6.3 yterms — *The y values of an array of vectors for the gradient* . . . 158
91.6.4 zterms — *The z values of an array of vectors for the gradient* . . . 159
91.7 methods . . . 159
91.7.1 make_point_terms — *A function to generate polynomial terms* . . . 159
91.7.2 make_ntgrl_terms — *A function to generate terms averaged over a box* . . . 160
91.7.3 make_gradient_terms — *A function to generate gradient vector terms* . . . 160
91.7.4 getTag — *Get tag value.* . . . 160
91.7.5 getSize — *Get total size.* . . . 160
91.7.6 getDimension — *Get number of dimensions.* . . . 161

92 StencilMatrix — *A matrix that relates polynomial coefficients to field values.* . . . 164
92.1 copy and assignment helper methods . . . 166
92.2 data . . . 166
92.3 methods . . . 167
92.3.1 fill_matrix — *Fill matrix based on field for a specific stencil.* . . . 167
92.4 usual methods . . . 164
92.5 data . . . 165
92.5.1 _col_by_col_matrix — *matrix of polynomial terms and sites.* . . . 165
92.6 methods. . . . 165
92.6.1 fill_matrix — *Fills-in the matrix according to a certain stencil.* . . . 166

93 ArbitraryMatrix — *A matrix that relates polynomial coefficients to field values.* . . . 168
93.1 copy and assignment helper methods . . . 171
93.2 data . . . 172
93.3 methods . . . 172
93.4 usual methods . . . 168
93.5 data . . . 170
93.5.1 _col_by_col_matrix — *matrix of polynomial terms and sites.* . . . 170
93.6 methods. . . . 170
93.6.1 getTag — *Get tag value.* . . . 171
93.6.2 numTerms — *Get number of terms.* . . . 171
93.6.3 numSites — *Get number of sites.* . . . 171

94 StencilVector — *An array of double precision numbers.* . . . 173
95 StencilVector_pointer . . . 174
96 StencilVector_ref . . . 175
97 StencilVector_const_pointer . . . 176

98 StencilVector_const_ref . . . 177
99 StencilVector_iterator . . . 178
100 StencilVector_const_iterator . . . 179
101 newStencilVector . . . 180
102 newStencilVector . . . 181
103 ImageBase — *Contains information that describes a field.* . . . 182
103.1 copy and assignment helper methods . . . 185
103.1.1 convert — *Copy of data members* . . . 185
103.1.2 convert_tree — *Call convert_tree on each parent class then call convert* . . . 186
103.2 data . . . 186
103.3 methods . . . 186
103.4 static constants . . . 183
103.5 usual methods . . . 183
103.6 virtual methods . . . 183
103.6.1 find_Indices_Nearest — *Finds nearest lattice pixel or voxel and displacement.* . . . 184
103.6.2 find_Indices_Nearest — *Finds nearest lattice pixel or voxel and displacement.* . . . 185
103.7 methods . . . 185

104 BasicDataType — *conversion from a type or character array to an integer descriptor* . . . 187
104.1 toDataType — *Conversion from a type an integer descriptor.* . . . 187
104.2 toDataType — *Conversion from a character array to an integer descriptor.* . . . 187

105 ObjVar — *ObjVar takes control of a pointer.* . . . 188
105.1 Methods. . . . 189
105.2 data . . . 189
105.3 methods . . . 188

106 LinTrans — *A linear transformation.* . . . 190
106.1 data . . . 190
106.2 copy and assignment helper methods . . . 191
106.3 usual methods . . . 190
106.4 methods . . . 191

107 MapDef — *Linear mapping between two fields.* . . . 192
107.1 data . . . 193
107.1.1 linear — *a linear transformation* . . . 193
107.1.2 beg_field_pos — *left justify, center, or right justify initial field* . . . 193
107.1.3 end_field_pos — *left justify, center, or right justify destination field* . . . 194
107.2 copy and assignment helper methods . . . 194
107.3 usual methods . . . 194
107.4 methods . . . 194

108 ImageFieldAssign — *Allows copying and assignment between different types.* . . . 195
108.1 type definitions . . . 196
108.2 access . . . 196
108.3 helper methods for copy and assignment between types . . . 202
108.4 copy and assignment helper methods . . . 202
108.5 usual methods . . . 197
108.6 access . . . 197
108.7 arithmetic methods . . . 197
108.8 static methods . . . 201
108.9 copy and assignment between different types . . . 201

108.10 I/O 201

109 operator<< — *ImageFieldAssign write to standard output.* 203
110 operator>> — *ImageFieldAssign read from standard input.* 204
111 ImageFieldBase — *Contains an array of field values.* 205
111.1 data 207
111.2 copy and assignment helper methods 207
111.3 usual methods 205
111.4 redefine some virtual functions of ImageBase 206
111.5 virtual methods 206
111.6 methods 206

112 ImageField — *Abstract class that declares image methods.* 208
112.1 constructors to pass information to LinAlgVector 208
112.2 virtual functions 209
112.2.1 new_by_interpol — *generate field by interpolation* 213

113 NewImageField — *Creates new ImageFieldTyped<T> pointers.* 214
113.1 static methods 214

114 ImageFieldTyped — *A basic field with a specified (templated) numerical type.* 215
114.1 type definitions 216
114.2 helper methods 226
114.3 copy and assignment helper methods 225
114.4 usual methods 216
114.5 methods 217
114.6 static methods 217
114.7 Arithmetic methods. 218
114.8 virtual functions of LinAlgVector 220
114.9 virtual functions of ImageField 221
114.10 templated methods 225

115 operator+ 227
116 operator- 228
117 operator* 229
118 operator+ 230
119 operator- 231
120 operator* 232
121 operator+ 233
122 operator- 234
123 operator* 235
124 operator+ 236
125 operator- 237
126 operator* 238
127 operator+ 239
128 operator- 240
129 operator* 241
130 operator+ 242
131 operator- 243
132 operator* 244
133 operator+ 245
134 operator- 246
135 operator* 247
136 operator+ 248
137 operator- 249

138 operator* 250
139 operator+ 251
140 operator- 252
141 operator* 253
142 operator+ 254
143 operator- 255
144 operator* 256
145 operator+ 257
146 operator- 258
147 operator* 259
148 operator+ 260
149 operator- 261
150 operator* 262
151 operator+ 263
152 operator- 264
153 operator* 265
154 operator+ 266
155 operator- 267
156 operator* 268
157 operator+ 269
158 operator- 270
159 operator* 271
160 operator<< — *ImageFieldTyped write to standard output.* 272
161 operator>> — *ImageFieldTyped read from standard input.* 273
162 ImageFieldAlgorithms — *Simple algorithms applied to a simple field* 274
162.1 static methods 274

163 FieldInterpolHelper — *helper methods for interpolation algorithms* 276
163.1 copy and assignment helper methods 276
163.2 data 277
163.3 initialization 277
163.4 usual methods 276

164 FieldInterpolAlgorithms — *interpolation algorithms for a regular grid* 278
164.1 static methods 278
164.1.1 new_by_interpol — *Constructor using a mapping of a grid.* 279
164.2 private static methods 279

165 RegridBrick — *Changes the resolution of an image.* 280
165.1 usual methods 281
165.2 static methods 281
165.3 new_brick — *Cannot change both aspect_ratio and lattice_bounds so this method is not public* 281

166 GridSlice — *Makes a two-dimensional grid for a slice of a three-dimensional grid.* 282
166.1 usual methods 282
166.2 methods 283
166.3 static methods 283
166.4 data 283

167 Documentation 284
167.1 Basic numbers and numerical vectors. 284
167.2 Stencils. 286

Class Graph 291

1 namespace **TNT**

In file ../tnt/vec.h:996503673

Names

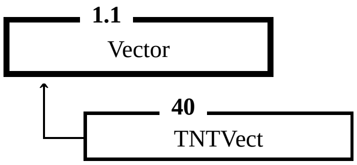
1.1 template<class T> class **Vector** *Basic TNT numerical vector.* 11

1.1 template<class T> class **Vector**

Basic TNT numerical vector.

In file ../tnt/vec.h:52

Inheritance



Basic TNT numerical vector.

Template Numerical Toolkit (TNT): Linear Algebra Module

Mathematical and Computational Sciences Division National Institute of Technology, Gaithersburg, MD USA

2 namespace **CastToSelfType**

Contains a method to cast a base class to a derived class

In file ../LinAlg/cast_to_self_type.hh:1595958638

Names

2.1 template<class T, class U> T&
cast_to_self_type (U& in) *Casts a base class U to a derived class T.* . . 12

Contains a method to cast a base class to a derived class

2.1 template<class T, class U> T& **cast_to_self_type** (U& in)

Casts a base class U to a derived class T.

In file ../LinAlg/cast_to_self_type.hh:446

Casts a base class U to a derived class T.

Requires that when the reference to the base class U is an actual instantation of the base class, the base contains a valid pointer to a derived class, which is returned by getBase().

Usage

Serves as a helper in defining the following functions for a derived class.

```
class Derived {

  inline static Derived&
  cast_to_self_type(Base& in) {
    return CastToSelfType::cast_to_self_type<Derived>(in);
  }

  inline static const Derived&
  cast_to_self_type(const Base& in) {
    return CastToSelfType::cast_to_self_type<const Derived>(in);
  }
}
```

Return Value: `reference` to derived class
Parameters: `in` reference to base class
Author: Alan Louis Scheinine
Version: `$Id: cast_to_self_type.hh,v 1.1 2002/04/03 19:54:33 alan Exp $`

3

int **invert_matrix** (double *matrix, int size, int ifdebug)

Inverts a matrix using LAPACK routines.

In file ../LinAlg/invert.hh:478

Inverts a matrix using LAPACK routines.

Return Value:	`0`	for success, negative for failure
Parameters:	`matrix`	the square matrix that is inverted
	`size`	the number of rows of the matrix
	`ifdebug`	a value of 1 activates debugging
Author:	Alan Louis Scheinine	
Version:	`$Id: invert.hh,v 1.1 2002/04/03 19:54:33 alan Exp $`	

4 template<class T> class **LimitRange**

Converts between various primitive number types.

In file ../LinAlg/limit_range.hh:520

Public Members

4.1 **methods** 14

Converts between various primitive number types.

Used primarily to avoid compiler warnings.

Author: Alan Louis Scheinine
Version: `$Id: limit_range.hh,v 1.1 2002/04/03 19:54:33 alan Exp $`

4.1 methods

Names

static	T	**limit_range** (char s)
static	T	**limit_range** (signed char s)
static	T	**limit_range** (unsigned char s)
static	T	**limit_range** (short s)
static	T	**limit_range** (unsigned short s)
static	T	**limit_range** (int s)
static	T	**limit_range** (unsigned int s)
static	T	**limit_range** (long s)
static	T	**limit_range** (unsigned long s)
static	T	**limit_range** (float s)
static	T	**limit_range** (double s)
static	T	**limit_range** (long double s)

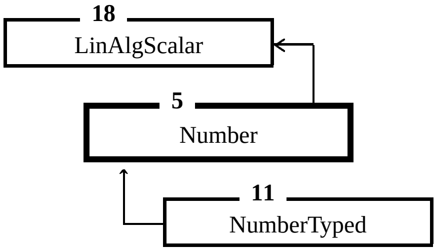
5

class **Number** : virtual public LinAlgScalar

Can represent any type of number.

In file ../LinAlg/number.hh:780

Inheritance



Private Members

5.1 **cast underlying rep to Number pointer** 15

Can represent any type of number.

Useful as parameter or return value of a virtual function.

One can have a Number* that is really a NumberTyped<T>* or one can have a Number* that is nothing more. In the latter case, the data member of LinAlgScalar, **lin_alg_scalar_**, points to a derived class such as NumberTyped<T>*. In the former case, the data member **rep** is null.

Note, each instantation has its own instantation of a LinAlgScalar* **lin_alg_scalar_**, so it is deleted when the object is deleted.

Author: Alan Louis Scheinine
Version: `$Id: number.hh,v 1.1 2002/04/03 19:54:33 alan Exp $`

5.1

cast underlying rep to Number pointer

Names

inline Number*
cast_to_self_type ()

inline const Number*
cast_to_self_type () const

5.1.1 **usual methods** 16

5.1.2 **virtual methods** 17

5.1.1

usual methods

Names

5.1.1.1 **Number** () *default constructor* 16

5.1.1.2 **Number** (const Number& object_in)
copy constructor 16

5.1.1.3 virtual Number&
operator= (const Number& object_in)
assignment 17

5.1.1.4 **˜Number** () *destructor* 17

5.1.1.1

Number ()

default constructor

In file ../LinAlg/number.hh:806

default constructor

5.1.1.2

Number (const Number& object_in)

copy constructor

In file ../LinAlg/number.hh:808

copy constructor

5.1.1.3

virtual Number& **operator=** (const Number& object_in)

assignment

In file ../LinAlg/number.hh:811

assignment

5.1.1.4

~Number ()

destructor

In file ../LinAlg/number.hh:816

destructor

5.1.2

virtual methods

Names

virtual	Number*	
		newNumber () const
virtual	Number*	
		cloneNumber () const
virtual	char	**getNumber_char** () const
virtual	signed char	
		getNumber_signed_char () const
virtual	unsigned char	
		getNumber_unsigned_char () const
virtual	short	**getNumber_short** () const
virtual	unsigned short	
		getNumber_unsigned_short () const
virtual	int	**getNumber_int** () const
virtual	unsigned int	
		getNumber_unsigned_int () const
virtual	long	**getNumber_long** () const
virtual	unsigned long	
		getNumber_unsigned_long () const
virtual	float	**getNumber_float** () const

virtual	double	**getNumber_double** () const
virtual	long double	**getNumber_long_double** () const
virtual	void	**setNumber** (char v)
virtual	void	**setNumber** (signed char v)
virtual	void	**setNumber** (unsigned char v)
virtual	void	**setNumber** (short v)
virtual	void	**setNumber** (unsigned short v)
virtual	void	**setNumber** (int v)
virtual	void	**setNumber** (unsigned int v)
virtual	void	**setNumber** (long v)
virtual	void	**setNumber** (unsigned long v)
virtual	void	**setNumber** (float v)
virtual	void	**setNumber** (double v)
virtual	void	**setNumber** (long double v)

6

inline Number **operator+** (const Number& A, const Number& B)

In file ../LinAlg/number.hh:963

7

inline Number **operator-** (const Number& A, const Number& B)

In file ../LinAlg/number.hh:967

8

inline Number **operator*** (const Number& A, const Number& B)

In file ../LinAlg/number.hh:971

9

inline Number **operator/** (const Number& A, const Number& B)

In file ../LinAlg/number.hh:975

10

namespace **GetNumber**

template parameterized getNumber()

In file ../LinAlg/number.hh:0

Names

10.1 template<class T> T
getNumber (const Number& n)
Gets a primitive number from wrapper class Number. 23

template parameterized getNumber()

10.1

template<class T> T **getNumber** (const Number& n)

Gets a primitive number from wrapper class Number.

In file ../LinAlg/number.hh:986

Gets a primitive number from wrapper class Number.

Return Value:	`a`	primitive of type T
Parameters:	`n`	base class of Number

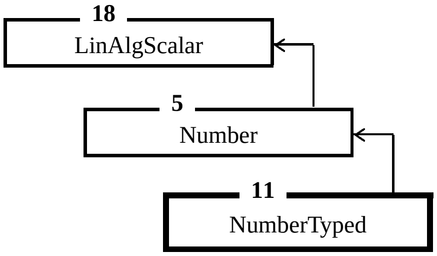
11

template<class T> class **NumberTyped** : public Number

Contains one scalar of type T.

In file ../LinAlg/number.hh:996

Inheritance



Public Members

11.1 **type definitions** 24
11.3 **usual methods** 25
11.4 **methods** 26
11.5 **arithmetic methods** 26
11.6 **helper methods** 27
11.7 **virtual methods of LinAlgScalar** 27
11.8 **virtual methods of Number** 28

Protected Members

11.2 **data** 29

Contains one scalar of type T.

Author: Alan Louis Scheinine
Version: `$Id: number.hh,v 1.1 2002/04/03 19:54:33 alan Exp $`

11.1

type definitions

Names

typedef T **value_type**

typedef T* **pointer**

typedef T& **reference**

typedef const T* **const_pointer**

typedef const T& **const_reference**

11.3 usual methods

Names

NumberTyped () *default constructor*

NumberTyped (char v) *constructor*

NumberTyped (signed char v) *constructor*

NumberTyped (unsigned char v) *constructor*

NumberTyped (short v) *constructor*

NumberTyped (unsigned short v) *constructor*

NumberTyped (int v) *constructor*

NumberTyped (unsigned int v) *constructor*

NumberTyped (long v) *constructor*

NumberTyped (unsigned long v) *constructor*

NumberTyped (float v) *constructor*

NumberTyped (double v)*constructor*

NumberTyped (long double v) *constructor*

NumberTyped (const NumberTyped<T> & x) *copy constructor*

NumberTyped (const Number& x) *constructor*

inline NumberTyped<T> & **operator=** (const NumberTyped<T> & x) *assignment*

inline NumberTyped<T> &

operator= (char v) *assignment*

inline NumberTyped<T> & **operator=** (signed char v) *assignment*

inline NumberTyped<T> & **operator=** (unsigned char v) *assignment*

inline NumberTyped<T> & **operator=** (short v) *assignment*

inline NumberTyped<T> & **operator=** (unsigned short v) *assignment*

inline NumberTyped<T> & **operator=** (int v) *assignment*

inline NumberTyped<T> & **operator=** (unsigned int v) *assignment*

inline NumberTyped<T> & **operator=** (long v) *assignment*

inline NumberTyped<T> & **operator=** (unsigned long v) *assignment*

inline NumberTyped<T> & **operator=** (float v) *assignment*

inline NumberTyped<T> & **operator=** (double v) *assignment*

inline NumberTyped<T> & **operator=** (long double v) *assignment*

virtual **~NumberTyped** () *destructor*

11.4 methods

Names

inline **operator const T&** () const

inline **operator T&** ()

11.5 arithmetic methods

Names

inline NumberTyped<T> &
operator+= (T d)

inline NumberTyped<T> &
operator-= (T d)

inline NumberTyped<T> &
operator*= (T d)

inline NumberTyped<T> &
operator/= (T d)

inline NumberTyped<T> &
operator+= (const NumberTyped<T> & d)

inline NumberTyped<T> &
operator-= (const NumberTyped<T> & d)

inline NumberTyped<T> &
operator*= (const NumberTyped<T> & d)

inline NumberTyped<T> &
operator/= (const NumberTyped<T> & d)

11.6 helper methods

Names

static NumberTyped<T> &
cast_to_self_type (LinAlgScalar& n)

static const NumberTyped<T> &
cast_to_self_type (const LinAlgScalar& n)

11.7 virtual methods of LinAlgScalar

Names

inline LinAlgScalar*
newLinAlgScalar () const

inline LinAlgScalar*
clone () const

inline LinAlgScalar&
operator= (const LinAlgScalar& las)

11.7.1 inline LinAlgScalar&

operator- () *A virtual function of class LinAlgScalar.* . . . 28

inline LinAlgScalar&
operator+= (const LinAlgScalar& las)

inline LinAlgScalar&
operator-= (const LinAlgScalar& las)

inline LinAlgScalar&
operator*= (const LinAlgScalar& las)

inline LinAlgScalar&
operator/= (const LinAlgScalar& las)

inline std::ostream&
put (std::ostream& s) const

inline std::istream&
get (std::istream& s)

11.7.1

inline LinAlgScalar& **operator-** ()

A virtual function of class LinAlgScalar.

In file ../LinAlg/number.hh:1230

A virtual function of class LinAlgScalar. Note, the following is not defined due to the error ”sorry, not implemented: adjusting pointers for covariant returns” inline NumberTyped<T> & operator-() return *this;

11.8

virtual methods of Number

Names

inline Number&
operator= (const Number& object_in)

inline NumberTyped<T> *
newNumber () const

inline Number*
cloneNumber () const

inline char **getNumber_char** () const

inline signed char
getNumber_signed_char () const

inline unsigned char
getNumber_unsigned_char () const

inline short **getNumber_short** () const inline unsigned short **getNumber_unsigned_short** () const

inline int **getNumber_int** () const

inline unsigned int **getNumber_unsigned_int** () const

inline long **getNumber_long** () const

inline unsigned long **getNumber_unsigned_long** () const

inline float **getNumber_float** () const

inline double **getNumber_double** () const

inline long double **getNumber_long_double** () const

inline void **setNumber** (char v)

inline void **setNumber** (signed char v)

inline void **setNumber** (unsigned char v)

inline void **setNumber** (short v)

inline void **setNumber** (unsigned short v)

inline void **setNumber** (int v)

inline void **setNumber** (unsigned int v)

inline void **setNumber** (long v)

inline void **setNumber** (unsigned long v)

inline void **setNumber** (float v)

inline void **setNumber** (double v)

inline void **setNumber** (long double v)

11.2

data

Names

T	**_value**	*one primitive numeric value*

12

template<class T>inline NumberTyped<T> **operator+** (const NumberTyped<T> & a, const NumberTyped<T> & b)

In file ../LinAlg/number.hh:1390

13

template<class T>inline NumberTyped<T> **operator-** (const NumberTyped<T> & a, const NumberTyped<T> & b)

In file ../LinAlg/number.hh:1395

14

template<class T>inline NumberTyped<T> **operator*** (const NumberTyped<T> & a, const NumberTyped<T> & b)

In file ../LinAlg/number.hh:1400

15

template<class T>inline NumberTyped<T> **operator/** (const NumberTyped<T> & a, const NumberTyped<T> & b)

In file ../LinAlg/number.hh:1405

16

std::ostream& **operator<<** (std::ostream& s, const LinAlgScalar& r)

In file ../LinAlg/lin_alg_vector.hh:1443

17

std::istream& **operator>>** (std::istream& s, LinAlgScalar& r)

In file ../LinAlg/lin_alg_vector.hh:1445

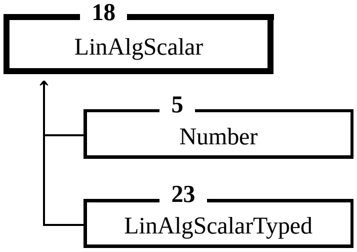
18

class **LinAlgScalar**

A scalar for linear algebra, independent of numeric type

In file ../LinAlg/lin_alg_vector.hh:1449

Inheritance



Public Members

18.2 **usual methods** 36

18.3 **virtual methods** 37

18.4 **public methods** 38

Protected Members

18.1 **data** 38

Private Members

friend std::ostream&
operator<< (std::ostream& s, const LinAlgScalar& r)

friend std::istream&
operator>> (std::istream& s, LinAlgScalar& r)

A scalar for linear algebra, independent of numeric type

18.2

usual methods

Names

		LinAlgScalar ()	
		LinAlgScalar (const LinAlgScalar& las)	
	virtual	LinAlgScalar& **operator=** (const LinAlgScalar& las)	
18.2.1	virtual	**˜LinAlgScalar** ()	*Each instantation has its own instantation of a LinAlgScalar* lin_alg_scalar_ so it is deleted when the object is deleted* 37

18.2.1

virtual **˜LinAlgScalar** ()

Each instantation has its own instantation of a LinAlgScalar lin_alg_scalar_ so it is deleted when the object is deleted*

In file ../LinAlg/lin_alg_vector.hh:1487

Each instantation has its own instantation of a LinAlgScalar* lin_alg_scalar_ so it is deleted when the object is deleted

18.3

virtual methods

Names

virtual LinAlgScalar*
newLinAlgScalar () const

virtual LinAlgScalar*
clone () const

virtual LinAlgScalar&
operator- ()

virtual LinAlgScalar&
operator+= (const LinAlgScalar& las)

virtual LinAlgScalar&
operator-= (const LinAlgScalar& las)

virtual LinAlgScalar&
operator*= (const LinAlgScalar& las)

virtual LinAlgScalar&
operator/= (const LinAlgScalar& las)

18.4

public methods

Names

inline LinAlgScalar*
getLinAlgScalar ()

inline const LinAlgScalar*
getLinAlgScalar () const

inline LinAlgScalar*
getBare ()

inline const LinAlgScalar*
getBare () const

virtual std::ostream&
put (std::ostream& s) const

virtual std::istream&
get (std::istream& s)

18.1

data

Names

18.1.1 LinAlgScalar* **lin_alg_scalar_** *Is non-zero only when the actual, highest level class is the base class* 38

18.1.1

LinAlgScalar* **lin_alg_scalar_**

Is non-zero only when the actual, highest level class is the base class

In file ../LinAlg/lin_alg_vector.hh:1464

Is non-zero only when the actual, highest level class is the base class

19

inline LinAlgScalar **operator+** (const LinAlgScalar& A, const LinAlgScalar& B)

In file ../LinAlg/lin_alg_vector.hh:1580

20

inline LinAlgScalar **operator-** (const LinAlgScalar& A, const LinAlgScalar& B)

In file ../LinAlg/lin_alg_vector.hh:1584

21

inline LinAlgScalar **operator*** (const LinAlgScalar& A, const LinAlgScalar& B)

In file ../LinAlg/lin_alg_vector.hh:1588

22

inline LinAlgScalar **operator/** (const LinAlgScalar& A, const LinAlgScalar& B)

In file ../LinAlg/lin_alg_vector.hh:1592

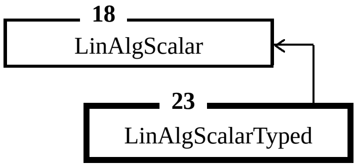
23

template<class T> class **LinAlgScalarTyped** : public LinAlgScalar

The class LinAlgScalarTyped<T> contains one scalar of type T

In file ../LinAlg/lin_alg_vector.hh:1599

Inheritance



Public Members

LinAlgScalarTyped does not use the pointer lin_alg_scalar_ in LinAl-

23.1 **type definitions** 44
23.3 **methods** 44

Protected Members

23.2 **data** 46

The class LinAlgScalarTyped<T> contains one scalar of type T

23.1 type definitions

Names

typedef T **value_type**

typedef T* **pointer**

typedef T& **reference**

typedef const T*
const_pointer

typedef const T&
const_reference

23.3 methods

Names

LinAlgScalarTyped () *default constructor*

LinAlgScalarTyped (const LinAlgScalarTyped<T> & x)
copy constructor

LinAlgScalarTyped (const LinAlgScalar& las)
constructor

LinAlgScalarTyped (T x)
constructor

inline LinAlgScalarTyped<T> &
operator= (const LinAlgScalarTyped<T> & x)
assignment

inline LinAlgScalarTyped<T> &
operator= (T d) *assignment*

virtual **~LinAlgScalarTyped** () *destructor*

static LinAlgScalarTyped<T> &
cast_to_self_type (LinAlgScalar& las)

static const LinAlgScalarTyped<T> &
cast_to_self_type (const LinAlgScalar& las)

inline LinAlgScalar&

operator= (const LinAlgScalar& las)
A virtual function of class LinAlgScalar.

inline LinAlgScalarTyped<T> *
newLinAlgScalar () const
A virtual function of class LinAlgScalar.

inline LinAlgScalarTyped<T> *
clone () const *A virtual function of class LinAlgScalar.*

inline **operator const T&** () const

inline **operator T&** ()

inline LinAlgScalarTyped<T> &
operator+= (T d)

inline LinAlgScalarTyped<T> &
operator-= (T d)

inline LinAlgScalarTyped<T> &
operator*= (T d)

inline LinAlgScalarTyped<T> &
operator/= (T d)

inline LinAlgScalarTyped<T> &
operator+= (const LinAlgScalarTyped<T> & las)

inline LinAlgScalarTyped<T> &
operator-= (const LinAlgScalarTyped<T> & las)

inline LinAlgScalarTyped<T> &
operator*= (const LinAlgScalarTyped<T> & las)

inline LinAlgScalarTyped<T> &
operator/= (const LinAlgScalarTyped<T> & las)

23.3.1 inline LinAlgScalar&
operator- () *A virtual function of class LinAlgScalar.* . . . 46

inline LinAlgScalar&
operator+= (const LinAlgScalar& las)
A virtual function of class LinAlgScalar.

inline LinAlgScalar&
operator-= (const LinAlgScalar& las)
A virtual function of class LinAlgScalar.

inline LinAlgScalar&
operator*= (const LinAlgScalar& las)
A virtual function of class LinAlgScalar.

inline LinAlgScalar&
operator/= (const LinAlgScalar& las)
A virtual function of class LinAlgScalar.

inline std::ostream&
put (std::ostream& s) const
A virtual function of class LinAlgScalar.

inline std::istream&
get (std::istream& s) *A virtual function of class LinAlgScalar.*

23.3.1

inline LinAlgScalar& **operator-** ()

A virtual function of class LinAlgScalar.

In file ../LinAlg/lin_alg_vector.hh:1718

A virtual function of class LinAlgScalar. Note, the following is not defined due to the error ”sorry, not implemented: adjusting pointers for covariant returns” inline LinAlgScalarTyped<T> & operator-() local_variable_ = -local_variable_; return *this;

23.2

data

Names

T	**local_variable_**

24

template<class T>inline LinAlgScalarTyped<T> **operator+** (const LinAlgScalarTyped<T> & a, const LinAlgScalarTyped<T> & b)

In file ../LinAlg/lin_alg_vector.hh:1807

25

template<class T>inline LinAlgScalarTyped<T> **operator-** (const LinAlgScalarTyped<T> & a, const LinAlgScalarTyped<T> & b)

In file ../LinAlg/lin_alg_vector.hh:1812

26

template<class T>inline LinAlgScalarTyped<T> **operator*** (const LinAlgScalarTyped<T> & a, const LinAlgScalarTyped<T> & b)

In file ../LinAlg/lin_alg_vector.hh:1817

27

template<class T>inline LinAlgScalarTyped<T> **operator/** (const LinAlgScalarTyped<T> & a, const LinAlgScalarTyped<T> & b)

In file ../LinAlg/lin_alg_vector.hh:1822

28

std::ostream& **operator<<** (std::ostream& s, const LinAlgVector& r)

In file ../LinAlg/lin_alg_vector.hh:1828

29

std::istream& **operator>>** (std::istream& s, LinAlgVector& r)

In file ../LinAlg/lin_alg_vector.hh:1830

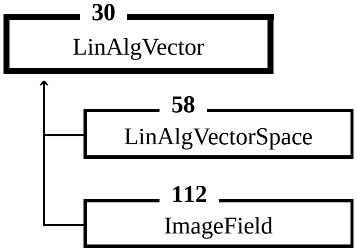
30

class **LinAlgVector**

Base class for a vector used in linear algebra

In file ../LinAlg/lin_alg_vector.hh:1834

Inheritance



Public Members

30.2 **usual methods** 53
30.3 **virtual methods** 54
30.4 **public methods** 55

Protected Members

30.1 **data** 56

Private Members

friend std::ostream&
operator<< (std::ostream& s, const LinAlgVector& r)

friend std::istream&
operator>> (std::istream& s, LinAlgVector& r)

Base class for a vector used in linear algebra

30.2

usual methods

Names

LinAlgVector ()

30.2.1 **LinAlgVector** (const LinAlgVector& lav)
Each instantation of LinAlgVector has its own lin_alg_vector_ , created using clone(). 54

virtual LinAlgVector&
operator= (const LinAlgVector& lav)

virtual **˜LinAlgVector** ()

30.2.1

LinAlgVector (const LinAlgVector& lav)

Each instantation of LinAlgVector has its own lin_alg_vector_ , created using clone().

In file ../LinAlg/lin_alg_vector.hh:1863

Each instantation of LinAlgVector has its own lin_alg_vector_ , created using clone(). The method clone() is a virtual function that constructs the actual class of the input.

30.3

virtual methods

Names

virtual LinAlgVector*
newLinAlgVector () const

virtual LinAlgVector*
clone () const

virtual LinAlgVector&
operator- ()

virtual LinAlgVector&
operator= (const LinAlgScalar& las)

virtual LinAlgVector&
operator+= (const LinAlgScalar& las)

virtual LinAlgVector&
operator-= (const LinAlgScalar& las)

virtual LinAlgVector&
operator*= (const LinAlgScalar& las)

virtual LinAlgVector&
operator/= (const LinAlgScalar& las)

virtual LinAlgVector&

operator= (const double& D)

virtual LinAlgVector&
operator+= (const double& D)

virtual LinAlgVector&
operator-= (const double& D)

virtual LinAlgVector&
operator*= (const double& D)

virtual LinAlgVector&
operator/= (const double& D)

virtual void **daxpy** (const LinAlgScalar& d, const LinAlgVector& lav)

virtual LinAlgScalar
Norm () const

virtual LinAlgScalar
Dot (const LinAlgVector& lav) const

virtual void **Orthog** ()

virtual LinAlgVector&
operator+= (const LinAlgVector& lav)

virtual LinAlgVector&
operator-= (const LinAlgVector& lav)

virtual LinAlgVector&
operator*= (const LinAlgVector& lav)

30.4 public methods

Names

inline LinAlgVector*
getLinAlgVector ()

inline const LinAlgVector*
getLinAlgVector () const

inline LinAlgVector*
getBare ()

inline const LinAlgVector*
getBare () const

virtual std::ostream&
put (std::ostream& s) const

virtual std::istream&
get (std::istream& s)

30.1 data

Names

30.1.1 LinAlgVector* **lin_alg_vector_** *Is non-zero only when the actual, highest level class is the base class* 56

30.1.1 LinAlgVector* **lin_alg_vector_**

Is non-zero only when the actual, highest level class is the base class

In file ../LinAlg/lin_alg_vector.hh:1849

Is non-zero only when the actual, highest level class is the base class

31

inline LinAlgVector **operator+** (const LinAlgScalar& las, const LinAlgVector& B)

In file ../LinAlg/lin_alg_vector.hh:2043

32

inline LinAlgVector **operator-** (const LinAlgScalar& las, const LinAlgVector& B)

In file ../LinAlg/lin_alg_vector.hh:2047

33

inline LinAlgVector **operator*** (const LinAlgScalar& las, const LinAlgVector& B)

In file ../LinAlg/lin_alg_vector.hh:2051

34

inline LinAlgVector **operator+** (const double& D, const LinAlgVector& B)

In file ../LinAlg/lin_alg_vector.hh:2055

35

inline LinAlgVector **operator-** (const double& D, const LinAlgVector& B)

In file ../LinAlg/lin_alg_vector.hh:2059

36

inline LinAlgVector **operator*** (const double& D, const LinAlgVector& B)

In file ../LinAlg/lin_alg_vector.hh:2063

37

inline LinAlgVector **operator+** (const LinAlgVector& A, const LinAlgVector& B)

In file ../LinAlg/lin_alg_vector.hh:2067

38

inline LinAlgVector **operator-** (const LinAlgVector& A, const LinAlgVector& B)

In file ../LinAlg/lin_alg_vector.hh:2071

39

inline LinAlgVector **operator*** (const LinAlgVector& A, const LinAlgVector& B)

In file ../LinAlg/lin_alg_vector.hh:2075

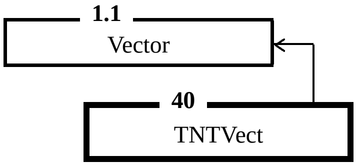
40

template<class T> class **TNTVect** : public Vector<T>

A TNT vector with some modifications.

In file ../LinAlg/tnt_vect.hh:2131

Inheritance



Public Members

40.3 **usual methods** 67

40.4 **methods of TNT Vector that return this** 68

40.5 **new methods not in TNT Vector** 68

40.6 **I/O** 68

Protected Members

40.1 **copy and assignment helper methods** 69

40.2 **fast copy and set** 69

Private Members

friend std::ostream&
operator<< **<T>** (std::ostream &s, const TNTVect<T> &A)

friend std::istream&
operator>> **<T>** (std::istream &s, TNTVect<T> &A)

A TNT vector with some modifications.

The template parameter, T, should be a number type, typically it is type **double**.

Global functions related to this class include

```
template <class T>
TNTVect<T> operator+(const TNTVect<T> &A,
     const TNTVect<T> &B)

template <class T>
TNTVect<T> operator-(const TNTVect<T> &A,
     const TNTVect<T> &B)

template <class T>
TNTVect<T> operator*(const TNTVect<T> &A,
     const TNTVect<T> &B)

template <class T>
T dot_prod(const TNTVect<T> &A,
   const TNTVect<T> &B)

template <class T>
std::ostream& operator<<(std::ostream &s, const TNTVect<T> &A)

template <class T>
std::istream& operator>>(std::istream &s, TNTVect<T> &A)
```

Author: Alan Louis Scheinine and Gassan Abdoulaev
Version: `$Id: tnt_vect.hh,v 1.2 2002/04/21 01:23:56 alan Exp $`

40.3 usual methods

Names

TNTVect () *default constructor*

TNTVect (const Vector<T> &A)
constructor

TNTVect (const TNTVect<T> &A)
copy constructor

TNTVect (Subscript N, const T& value = T(0))
constructor

TNTVect (Subscript N, const T* v)
constructor

TNTVect (Subscript N, char *s)
constructor

TNTVect<T> &
operator= (const TNTVect<T> &object_in)
assignment

TNTVect<T> &
operator= (const T& scalar)
assignment from scalar

~TNTVect () *destructor*

40.4 methods of TNT Vector that return this

Names

inline TNTVect<T> &
newsize (Subscript N)

40.5 new methods not in TNT Vector

Names

inline TNTVect<T> &
operator+= (const TNTVect<T> &A)

inline TNTVect<T> &
operator-= (const TNTVect<T> &A)

inline TNTVect<T> &
operator*= (const TNTVect<T> &A)

inline TNTVect<T> &
operator/= (const TNTVect<T> &A)

inline TNTVect<T> &
operator+= (const T& scalar)

inline TNTVect<T> &
operator-= (const T& scalar)

inline TNTVect<T> &
operator*= (const T& scalar)

inline void **daxpy_impl** (const T& scalar, const TNTVect<T> &A)

inline void **accumulate** (const_iterator beg, const_iterator end, T& s)

inline void **Orthog_impl** ()

40.6 I/O

Names

inline std::ostream&
put (std::ostream& s) const

inline std::istream&

get (std::istream& s)

40.1

copy and assignment helper methods

Names

40.1.1 inline void **convert** (const TNTVect<T>& object_in)
Copy of data members 69

40.1.1

inline void **convert** (const TNTVect<T>& object_in)

Copy of data members

In file ../LinAlg/tnt_vect.hh:2148

Copy of data members

40.2

fast copy and set

Names

inline void **copy** (const T* v)

inline void **set** (const T& val)

41

template<class T> TNTVect<T> **operator+** (const TNTVect<T> &A, const TNTVect<T> &B)

TNTVect sum.

In file ../LinAlg/tnt_vect.hh:2381

TNTVect sum. A method of TNT Vector converted to TNTVect.

42

template<class T> TNTVect<T> **operator-** (const TNTVect<T> &A, const TNTVect<T> &B)

TNTVect difference.

In file ../LinAlg/tnt_vect.hh:2395

TNTVect difference. A method of TNT Vector converted to TNTVect.

43

template<class T> TNTVect<T> **operator*** (const TNTVect<T> &A, const TNTVect<T> &B)

TNTVect component by component product.

In file ../LinAlg/tnt_vect.hh:2409

TNTVect component by component product. A method of TNT Vector converted to TNTVect.

44

template<class T> T **dot_prod** (const TNTVect<T> &A, const TNTVect<T> &B)

TNTVect inner product.

In file ../LinAlg/tnt_vect.hh:2423

TNTVect inner product. A method of TNT Vector converted to TNTVect.

45

template<class T> std::ostream& **operator<<** (std::ostream &s, const TNTVect<T> &A)

TNTVect write to standard output.

In file ../LinAlg/tnt_vect.hh:2437

TNTVect write to standard output. A method of TNT Vector converted to TNTVect.

46

template<class T> std::istream& **operator>>** (std::istream &s, TNTVect<T> &A)

TNTVect read from standard input.

In file ../LinAlg/tnt_vect.hh:2443

TNTVect read from standard input. A method of TNT Vector converted to TNTVect.

47

template<class T> class **ReadOnlyNumArray**

An array that can be declared Read Only

In file ../LinAlg/read_only_num_array.hh:2455

Public Members

47.2 **usual methods** 76

47.3 **methods** 77

Private Members

47.1 **data** 77

An array that can be declared Read Only

47.2

usual methods

Names

ReadOnlyNumArray () *default constructor*

ReadOnlyNumArray (int size_in)
constructor

ReadOnlyNumArray (const ReadOnlyNumArray<T>& object_in)
copy constructor

ReadOnlyNumArray<T> &
operator= (const ReadOnlyNumArray<T>& object_in)
assignment

ReadOnlyNumArray<T> &
operator= (const TNTVect<T>& object_in)
assignment

ReadOnlyNumArray<T> &
operator= (const T& scalar)
assignment from scalar

virtual **˜ReadOnlyNumArray** () *destructor*

47.3

methods

Names

inline	void	**setRO** ()
inline	bool	**getRO** () const
inline	void	**newsize** ()
inline	void	**newsize** (int size_in)
inline	int	**size** () const
inline	const T&	
		operator[] (int i) const
inline	void	**setValues** (int num_elem, const T* v)

47.1

data

Names

TNTVect<T>	**_v**
bool	**_ro**

48 class **Timer**

A stopwatch

In file ../LinAlg/timer.hh:2554

Public Members

48.2	**public typedef and data**	. . . .	78
48.3	**usual methods**	. . . .	78
48.4	**methods**	. . . .	79

Private Members

48.1	**data**	. . . .	79

A stopwatch

48.2 public typedef and data

Names

typedef clock_t **Clocks**

static const int **CPS**

48.3 usual methods

Names

Timer () *default constructor*

48.4
methods

Names

void	**start** ()	*starts the chronometer*
void	**stop** ()	*ends the chronometer*
void	**stop_cycle** ()	*stores a cycle and resets accumulator*
void	**reset** ()	*resets total*
double	**get_total** ()	*computes the time spent between end and start: time in seconds*
double	**get_avg** ()	*computes the time spent between end and start: time in seconds*

48.1
data

Names

clock_t	**start_**
int	**count_**
bool	**running**

49

ostream& **operator<<** (ostream& s, const Vector1& A)

In file ../LinAlg/vector123.hh:2662

50

istream& **operator>>** (istream& s, Vector1& A)

In file ../LinAlg/vector123.hh:2664

51

ostream& **operator<<** (ostream& s, const Vector2& A)

In file ../LinAlg/vector123.hh:2666

52

istream& **operator>>** (istream& s, Vector2& A)

In file ../LinAlg/vector123.hh:2668

53

ostream& **operator<<** (ostream& s, const Vector3& A)

In file ../LinAlg/vector123.hh:2670

54

istream& **operator>>** (istream& s, Vector3& A)

In file ../LinAlg/vector123.hh:2672

55

class **Vector1**

Include access by functions used by Vertex in order to generalize algorithms

In file ../LinAlg/vector123.hh:2678

Public Members

55.1 **public methods** 86

Private Members

friend ostream&
operator<< (ostream& s, const Vector1& A)

friend istream&**operator>>** (istream& s, Vector1& A)

Include access by functions used by Vertex in order to generalize algorithms

55.1

public methods

Names

double		**x**
inline		**Vector1** (double xin = 0.0, double yin = 0.0, double zin=0.0)
inline	double	**getXYZ** (int i) const
inline	void	**putXYZ** (int i, double d)
inline	double&	**X** ()
inline	const double&	**X** () const
		Vector1 (const Vector2& in)
		Vector1 (const Vector3& in)
inline	size_t	**size** () const

56 class **Vector2**

Include access by functions used by Vertex in order to generalize algorithms

In file ../LinAlg/vector123.hh:2721

Public Members

56.1 **public methods** 87

Private Members

friend ostream& **operator<<** (ostream& s, const Vector2& A)

friend istream&**operator>>** (istream& s, Vector2& A)

Include access by functions used by Vertex in order to generalize algorithms

56.1 public methods

Names

double **x**

inline **Vector2** (double xin = 0.0, double yin = 0.0, double zin=0.0)

inline double **getXYZ** (int i) const

inline void **putXYZ** (int i, double d)

inline double& **X** ()

inline const double& **X** () const

inline double& **Y** ()

inline const double& **Y** () const

Vector2 (const Vector1& in)

Vector2 (const Vector3& in)

inline size_t **size** () const

57

class **Vector3**

Include access by functions used by Vertex in order to generalize algorithms

In file ../LinAlg/vector123.hh:2769

Public Members

57.1 **public methods** 88

Private Members

friend ostream&
operator<< (ostream& s, const Vector3& A)

friend istream&**operator>>** (istream& s, Vector3& A)

Include access by functions used by Vertex in order to generalize algorithms

57.1

public methods

Names

double **x**

inline **Vector3** (double xin = 0.0, double yin = 0.0, double zin = 0.0)

inline double **getXYZ** (int i) const

inline void **putXYZ** (int i, double d)

inline double&
X ()

inline const double&
X () const

inline double&
Y ()

inline const double&
Y () const

inline double&
Z ()

inline const double&
Z () const

Vector3 (const Vector1& in)

Vector3 (const Vector2& in)

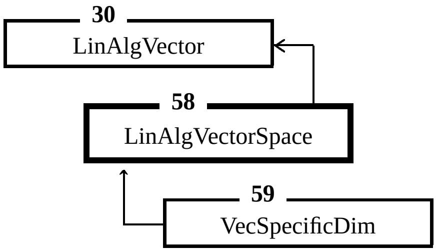
inline size_t **size** () const

58

class **LinAlgVectorSpace** : virtual public LinAlgVector

In file ../LinAlg/vector123.hh:2818

Inheritance



Public Members

58.2 **usual methods** 90

58.3 **virtual methods** 90

Private Members

58.1 **casting to actual type** 91

58.2

usual methods

Names

LinAlgVectorSpace ()

LinAlgVectorSpace (const LinAlgVectorSpace& lav)

LinAlgVectorSpace&
operator= (const LinAlgVectorSpace& lav)

~LinAlgVectorSpace ()

58.3

virtual methods

Names

virtual double **getXYZ** (int i) const

virtual void **putXYZ** (int i, double d)

virtual LinAlgVectorSpace&
Cross (const LinAlgVectorSpace& A, const LinAlgVectorSpace& B)

virtual void **normalize** ()

virtual double **NormSqd** () const

58.1

casting to actual type

Names

inline LinAlgVectorSpace*
cast_to_self_type ()

inline const LinAlgVectorSpace*
cast_to_self_type () const

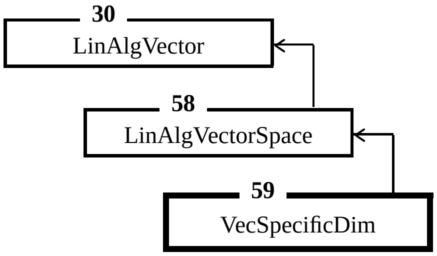
59

template<class T> class **VecSpecificDim** : virtual public LinAlgVectorSpace

T can be Vector1, Vector2, or Vector3

In file ../LinAlg/vector123.hh:2910

Inheritance



Public Members

59.2 **usual methods** 93

59.3 **static methods** 93

59.4 **access to data** 94

59.5 **operators** 94

59.6 **virtual functions of LinAlgVector** 94

59.7 **virtual functions of LinAlgVectorSpace** 95

59.8 **linear algebra functions** 95

59.9 **virtual functions of LinAlgVector** 96

Protected Members

59.1 **data** 96

Private Members

friend std::ostream&
operator<< <T> (std::ostream& s, const VecSpecificDim<T>& A)

friend std::istream&
operator>> <T> (std::istream& s, VecSpecificDim<T>& A)

T can be Vector1, Vector2, or Vector3

59.2 usual methods

Names

59.2.1 **VecSpecificDim** (double xin = 0.0, double yin = 0.0, double zin = 0.0)
default constructor 93

VecSpecificDim (T& in) *constructor*

VecSpecificDim (LinAlgVector& in)
constructor

VecSpecificDim (const VecSpecificDim<T>& in)
copy constructor

VecSpecificDim<T> &
operator= (const VecSpecificDim<T>& in)
assignment operator

˜VecSpecificDim () *destructor*

59.2.1 **VecSpecificDim** (double xin = 0.0, double yin = 0.0, double zin = 0.0)

default constructor

In file ../LinAlg/vector123.hh:2937

default constructor

Note, constructor of Vector1 and Vector2 will also accept three arguments.

59.3 static methods

Names

inline static VecSpecificDim<T> &
cast_to_self_type (LinAlgVector& in)

inline static const VecSpecificDim<T> &
cast_to_self_type (const LinAlgVector& in)

59.4

access to data

Names

inline **operator const T&** () const

inline **operator T&** ()

59.5

operators

Names

inline VecSpecificDim<T> &
operator+= (const VecSpecificDim<T>& A)

inline VecSpecificDim<T> &
operator-= (const VecSpecificDim<T>& A)

inline VecSpecificDim<T> &
operator*= (const VecSpecificDim<T>& A)

59.6

virtual functions of LinAlgVector

Names

inline LinAlgVector&
operator= (const LinAlgVector& in)

inline LinAlgVector*
newLinAlgVector () const

inline LinAlgVector*
clone () const

inline LinAlgVector&
operator- ()

inline LinAlgVector&
operator= (const LinAlgScalar& las)

inline LinAlgVector&
operator+= (const LinAlgScalar& las)

inline LinAlgVector&
operator-= (const LinAlgScalar& las)

inline LinAlgVector&

operator*= (const LinAlgScalar& las)

inline LinAlgVector&
operator/= (const LinAlgScalar& las)

inline LinAlgVector&
operator+= (const LinAlgVector& lav)

inline LinAlgVector&
operator-= (const LinAlgVector& lav)

inline LinAlgVector&
operator*= (const LinAlgVector& lav)

inline std::ostream&
put (std::ostream& s) const

inline std::istream&
get (std::istream& s)

59.7

virtual functions of LinAlgVectorSpace

Names

inline double **getXYZ** (int i) const

inline void **putXYZ** (int i, double d)

inline LinAlgVectorSpace&
Cross (const LinAlgVectorSpace& A, const LinAlgVectorSpace& B)

inline void **normalize** ()

inline double **NormSqd** () const

59.8

linear algebra functions

Names

inline void **daxpy_impl** (const double& scalar, const VecSpecificDim<T> &A)

inline void **Orthog_impl** ()

inline double **norm** () const

inline double **dot** (const VecSpecificDim<T>& v) const

inline double **norm_sqd** () const

59.9

virtual functions of LinAlgVector

Names

inline	void	**daxpy** (const LinAlgScalar& las, const LinAlgVector& lav)
inline	void	**Orthog** ()
inline	LinAlgScalar	**Norm** () const
inline	LinAlgScalar	**Dot** (const LinAlgVector& lav) const
inline	LinAlgVector&	**operator=** (const double& A)
inline	LinAlgVector&	**operator+=** (const double& A)
inline	LinAlgVector&	**operator-=** (const double& A)
inline	LinAlgVector&	**operator*=** (const double& A)
inline	LinAlgVector&	**operator/=** (const double& A)

59.1

data

Names

T	**vec_**

60

template<class T>inline LinAlgVector **operator+** (const double& A, const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3221

61

template<class T>inline LinAlgVector **operator-** (const double& A, const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3228

62

template<class T>inline LinAlgVector **operator*** (const double& A, const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3236

63

template<class T>inline LinAlgVector **operator+** (const LinAlgScalar& las, const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3243

64

template<class T>inline LinAlgVector **operator-** (const LinAlgScalar& las, const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3248

65

template<class T>inline LinAlgVector **operator*** (const LinAlgScalar& las, const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3253

66

template<class T>inline LinAlgVector **operator+** (const VecSpecificDim<T>& A, const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3258

67

template<class T>inline LinAlgVector **operator-** (const VecSpecificDim<T>& A, const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3265

68

template<class T>inline LinAlgVector **operator*** (const VecSpecificDim<T>& A, const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3272

69

template<> std::ostream& **operator<< <Vector1>** (std::ostream& s, const VecSpecificDim<Vector1>& A)

In file ../LinAlg/vector123.hh:3279

70

template<> std::istream& **operator>> <Vector1>** (std::istream& s, VecSpecificDim<Vector1>& A)

In file ../LinAlg/vector123.hh:3283

71

template**<>** std::ostream& **operator<< <Vector2>** (std::ostream& s, const VecSpecificDim<Vector2>& A)

In file ../LinAlg/vector123.hh:3287

72

template<> std::istream& **operator>> <Vector2>** (std::istream& s, VecSpecificDim<Vector2>& A)

In file ../LinAlg/vector123.hh:3291

73

template**<>** std::ostream& **operator<< <Vector3>** (std::ostream& s, const VecSpecificDim<Vector3>& A)

In file ../LinAlg/vector123.hh:3295

74

template**<>** std::istream& **operator>> <Vector3>** (std::istream& s, VecSpecificDim<Vector3>& A)

In file ../LinAlg/vector123.hh:3299

75 class **BsplineEquations**

B splines for interpolation.

In file ../Basic/bspline.hh:3336

Public Members

75.1 **basic equations** 112

B splines for interpolation.

$$B^n(x) = \sum_{k=0}^{n+1} \frac{(-1)^k (n+1)}{(n+1-k)!k!} (\frac{n+1}{2} + x - k)_+^n$$

$$\frac{d}{dx} B^n(x) = B^{n-1}(x+1/2) - B^{n-1}(x-1/2)$$

Author: Alan Louis Scheinine
Version: `$Id: bspline.hh,v 1.4 2002/04/17 15:44:07 alan Exp $`

75.1 basic equations

Names

static inline double
bspline2 (double x)

static inline double
bspline2_derivative (double x)

static inline double
bspline2_integral (double x, double y)

static inline double
bspline2_integral (double x)

75.1.1 static inline double
bsplinepair2 (double x) *input x is centered between two peaks* 114

75.1.2 static inline double
bsplinepair2_derivative (double x)
input x is centered between two peaks 114

75.1.3 static inline double
bsplinepair2_integral (double x, double y)
input x is centered between two peaks 114

75.1.4 static inline double

bsplinepair2_integral (double x)
input x is centered between two peaks 114

static inline double
bspline3 (double x)

static inline double
bspline3_derivative (double x)

static inline double
bspline3_integral (double x, double y)

static inline double
bspline3_integral (double x)

75.1.5 static inline double
bsplinepair3 (double x) *input x is centered between two peaks* 114

75.1.6 static inline double
bsplinepair3_derivative (double x)
input x is centered between two peaks 115

75.1.7 static inline double
bsplinepair3_integral (double x, double y)
input x is centered between two peaks 115

75.1.8 static inline double
bsplinepair3_integral (double x)
input x is centered between two peaks 115

static inline double
bspline4 (double x)

static inline double
bspline4_derivative (double x)

static inline double
bspline4_integral (double x, double y)

static inline double
bspline4_integral (double x)

75.1.9 static inline double
bsplinepair4 (double x) *input x is centered between two peaks* 115

75.1.10 static inline double
bsplinepair4_derivative (double x)
input x is centered between two peaks 116

75.1.11 static inline double
bsplinepair4_integral (double x, double y)
input x is centered between two peaks 116

75.1.12 static inline double
bsplinepair4_integral (double x)
input x is centered between two peaks 116

75.1.1

static inline double **bsplinepair2** (double x) *input x is centered between two peaks*

In file ../Basic/bspline.hh:3412

input x is centered between two peaks

75.1.2

static inline double **bsplinepair2_derivative** (double x)

input x is centered between two peaks

In file ../Basic/bspline.hh:3415

input x is centered between two peaks

75.1.3

static inline double **bsplinepair2_integral** (double x, double y)

input x is centered between two peaks

In file ../Basic/bspline.hh:3418

input x is centered between two peaks

75.1.4

static inline double **bsplinepair2_integral** (double x)

input x is centered between two peaks

In file ../Basic/bspline.hh:3421

input x is centered between two peaks

75.1.5

static inline double **bsplinepair3** (double x)

input x is centered between two peaks

In file ../Basic/bspline.hh:3496

input x is centered between two peaks

75.1.6

```
static inline   double bsplinepair3_derivative (double x)
```

input x is centered between two peaks

In file ../Basic/bspline.hh:3499

input x is centered between two peaks

75.1.7

```
static inline   double bsplinepair3_integral (double x, double y)
```

input x is centered between two peaks

In file ../Basic/bspline.hh:3502

input x is centered between two peaks

75.1.8

```
static inline   double bsplinepair3_integral (double x)
```

input x is centered between two peaks

In file ../Basic/bspline.hh:3505

input x is centered between two peaks

75.1.9

```
static inline   double bsplinepair4 (double x)
```

input x is centered between two peaks

In file ../Basic/bspline.hh:3633

input x is centered between two peaks

75.1.10

static inline double **bsplinepair4_derivative** (double x)

input x is centered between two peaks

In file ../Basic/bspline.hh:3636

input x is centered between two peaks

75.1.11

static inline double **bsplinepair4_integral** (double x, double y)

input x is centered between two peaks

In file ../Basic/bspline.hh:3639

input x is centered between two peaks

75.1.12

static inline double **bsplinepair4_integral** (double x)

input x is centered between two peaks

In file ../Basic/bspline.hh:3642

input x is centered between two peaks

76

class **Bspline**

B-spline with specific width.

In file ../Basic/bspline.hh:3668

Public Members

76.4 **usual methods** 117
76.5 **methods** 118

Protected Members

76.2 **copy and assignment helper methods** 118
76.3 **methods** 119

Private Members

76.1 **basic equations** 120

B-spline with specific width.

Author: Alan Louis Scheinine
Version: `$Id: bspline.hh,v 1.4 2002/04/17 15:44:07 alan Exp $`

76.4

usual methods

Names

76.4.1 **Bspline** () *Default constructor.* 118

Bspline (double x, int ix) *constructor*

Bspline (double x, double y, int ix, int iy)
constructor

Bspline (double x, double y, double z, int ix, int iy, int iz)
constructor

Bspline (const Bspline& object_in)
copy constructor

Bspline& **operator=** (const Bspline& object_in)

		assignment operator
virtual	**˜Bspline** ()	*destructor*

76.4.1

Bspline ()

Default constructor.

In file ../Basic/bspline.hh:3787

Default constructor. Not really meaningful.

76.5

methods

Names

inline	void	**getVoxSize** (double* x, double* y, double* z) const
inline	void	**getCenter** (double* x, double* y, double* z) const
inline	void	**setCenter** (double x, double y, double z)
inline	unsigned char	**getDimen** () const
inline	void	**setExtended** (signed char x, signed char y, signed char z)
inline	void	**getExtended** (signed char* x, signed char* y, signed char* z) const
inline	double	**loc_map** (int i, double a) const
inline	double	**loc_map_pair** (int i, double a) const
inline	double	**bspline** (const double* location) const
inline	void	**bspline_derivative** (const double* location, double* d) const
inline	double	**bspline_integral** (const double* box) const
inline	double	**bspline_avg** (const double* box) const

76.2

copy and assignment helper methods

Names

76.2.1 inline void **convert** (const Bspline& object_in)
Copy of data members 119

76.2.2 inline void **convert_tree** (const Bspline& object_in)
Call convert_tree on each parent class then call convert 119

76.2.1

inline void **convert** (const Bspline& object_in)

Copy of data members

In file ../Basic/bspline.hh:3696

Copy of data members

76.2.2

inline void **convert_tree** (const Bspline& object_in)

Call convert_tree on each parent class then call convert

In file ../Basic/bspline.hh:3714

Call convert_tree on each parent class then call convert

76.3 methods

Names

inline void **create_invr_scale** ()

inline void **check_degree** ()

76.1

basic equations

Names

unsigned char	**dimen_**
double	**vox_size_** [3]
double	**center_** [3]
double	**invr_scale_** [3]
unsigned char	**degree_** [3]
signed char	**extended_** [3]

77

template<class T> class **StencilHandle**

StencilHandle takes control of a pointer.

In file ../Basic/stencil_handle.hh:4124

Public Members

77.3 **methods** 121

Private Members

77.1 **methods** 122
77.2 **data** 122

StencilHandle takes control of a pointer.

A counted pointer that takes control of a pointer.

Modified from omniORB2 objectAdapter.h Created on: 5/3/99 Author : David Riddoch (djr) Copyright (C) 1996, 1999 AT&T Research Cambridge This file is part of the omniORB library. The omniORB library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

The counted pointer with a counter from the heap is shown in a Stroustrup book on C++.

Author: Alan Louis Scheinine
Version: `$Id: stencil_handle.hh,v 1.2 2002/04/22 05:24:20 alan Exp $`

77.3

methods

Names

StencilHandle ()	*default constructor*
StencilHandle (T* p)	*Constructor, now the StencilHandle is responsible for deleting T* p.*
StencilHandle (const StencilHandle<T>& r)	*copy constructor*
˜StencilHandle ()	*destructor*

StencilHandle<T> &

	operator= (T* p)	*Assignment, now the StencilHandle is responsible for deleting T* p.*
StencilHandle<T> &	**operator=** (const StencilHandle<T>& r)	*assignment*
inline T*	**operator->** () const	
inline	**operator T*** () const	
inline	**operator T*&** ()	

77.1 methods

Names

void	**release_ptr** ()	*reduce the reference count and maybe delete pointer and counter*
void	**duplicate_ptr** (const StencilHandle<T>& r)	*Become another carrier of the pointer.*

77.2 data

Names

T*	**ptr_**
int*	**reference_count_**

78

extern const int **POINT_VALUE_MODE**

Value set in stencil_matrix.C

In file ../Basic/interpol.hh:4226

79

extern const int **BOX_VALUE_MODE**

Value set in stencil_matrix.C

In file ../Basic/interpol.hh:4228

80

extern const int **PRECISION_LEVEL1**

Value set in Field/field_interpol_algorithms.C

In file ../Basic/interpol.hh:4231

81

extern const int **PRECISION_LEVEL2**

Value set in Field/field_interpol_algorithms.C

In file ../Basic/interpol.hh:4233

82

extern const int **PRECISION_LEVEL3**

Value set in Field/field_interpol_algorithms.C

In file ../Basic/interpol.hh:4235

83

extern const int **PRECISION_LEVEL4**

Value set in Field/field_interpol_algorithms.C

In file ../Basic/interpol.hh:4237

84

LinAlg read_only_num_array.hh const int **MAX_STENCIL_SITES**

In file ../Basic/stencil_sites.hh:4291

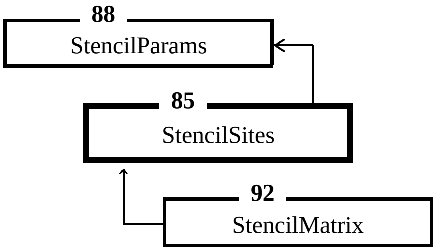
85

class **StencilSites** : public StencilParams

A stencil of sites.

In file ../Basic/stencil_sites.hh:4302

Inheritance



Public Members

85.6 **constructors** 131
85.7 **usual methods** 131
85.8 **Data** 131
85.9 **static methods** 133

Protected Members

85.2 **copy and assignment helper methods** 133
85.3 **methods** 134
85.4 **static methods** 134
85.5 **data** 134

Private Members

85.1 **methods** 136

A stencil of sites.

Note, assumes that the stencil tag is equal to the size of the stencil. There is no case in which different stencil patterns have the same size.

Author: Alan Louis Scheinine
Version: `$Id: stencil_sites.hh,v 1.5 2002/04/03 22:01:43 alan Exp $`

85.6

constructors

Names

85.6.1	**StencilSites** ()	*default constructor*	131
	StencilSites (int tag_in)	*Sets the x, y, [z] sites of a stencil.*	
	StencilSites (const StencilSitesTag& tag_in)	*Sets the x, y, [z] sites of a stencil.*	

85.6.1

StencilSites ()

default constructor

In file ../Basic/stencil_sites.hh:4556

default constructor

The default constructor is not really useful aside from implicit usage such as filling a vector with T().

85.7

usual methods

Names

	StencilSites (const StencilSites& object_in)	*Copy constructor.*
StencilSites&	**operator=** (const StencilSites& object_in)	*Assignment operator.*
virtual	**˜StencilSites** ()	*Destructor.*

85.8

Data

Names

85.8.1 ReadOnlyNumArray<signed char> **stencil_sites_x** *X positions of the stencil sites* 132

85.8.2 ReadOnlyNumArray<signed char> **stencil_sites_y** *Y positions of the stencil sites* 132

85.8.3 ReadOnlyNumArray<signed char> **stencil_sites_z** *Z positions of the stencil sites* 132

85.8.1

ReadOnlyNumArray<signed char> **stencil_sites_x**

X positions of the stencil sites

In file ../Basic/stencil_sites.hh:4606

X positions of the stencil sites

85.8.2

ReadOnlyNumArray<signed char> **stencil_sites_y**

Y positions of the stencil sites

In file ../Basic/stencil_sites.hh:4611

Y positions of the stencil sites

85.8.3

ReadOnlyNumArray<signed char> **stencil_sites_z**

Z positions of the stencil sites

In file ../Basic/stencil_sites.hh:4616

Z positions of the stencil sites

85.9

static methods

Names

static int **get_extent** (int tag_in)

85.2

copy and assignment helper methods

Names

85.2.1 inline void **convert** (const StencilSites& object_in)
Copy of data members 133

85.2.2 inline void **convert_tree** (const StencilSites& object_in)
Call convert_tree on each parent class then call convert 133

85.2.1

inline void **convert** (const StencilSites& object_in)

Copy of data members

In file ../Basic/stencil_sites.hh:4323

Copy of data members

85.2.2

inline void **convert_tree** (const StencilSites& object_in)

Call convert_tree on each parent class then call convert

In file ../Basic/stencil_sites.hh:4329

Call convert_tree on each parent class then call convert

85.3 methods

Names

inline void **set_stencil_sites_aux** (int size_in, const signed char* stencil_sites_xx,
const signed char* stencil_sites_yy,
const signed char* stencil_sites_zz)

void **set_stencil_sites** (int tag_in)

85.4 static methods

Names

static void **make_stencil_sites** ()

static void **make_stencil_sites0** ()

static void **make_stencil_sites3** ()

static void **make_stencil_sites5** ()

static void **make_stencil_sites9** ()

static void **make_stencil_sites13** ()

static void **make_stencil_sites21** ()

static void **make_stencil_sites25** ()

static void **make_stencil_sites27** ()

static void **make_stencil_sites33** ()

static void **make_stencil_sites57** ()

85.5 data

Names

static signed char
stencil_sites_x0 [MAX_STENCIL_SITES]

static signed char
stencil_sites_y0 [MAX_STENCIL_SITES]

static signed char
stencil_sites_z0 [MAX_STENCIL_SITES]

static signed char

stencil_sites_x3 [MAX_STENCIL_SITES]

static signed char
stencil_sites_y3 [MAX_STENCIL_SITES]

static signed char
stencil_sites_z3 [MAX_STENCIL_SITES]

static signed char
stencil_sites_x5 [MAX_STENCIL_SITES]

static signed char
stencil_sites_y5 [MAX_STENCIL_SITES]

static signed char
stencil_sites_z5 [MAX_STENCIL_SITES]

static signed char
stencil_sites_x9 [MAX_STENCIL_SITES]

static signed char
stencil_sites_y9 [MAX_STENCIL_SITES]

static signed char
stencil_sites_z9 [MAX_STENCIL_SITES]

static signed char
stencil_sites_x13 [MAX_STENCIL_SITES]

static signed char
stencil_sites_y13 [MAX_STENCIL_SITES]

static signed char
stencil_sites_z13 [MAX_STENCIL_SITES]

static signed char
stencil_sites_x21 [MAX_STENCIL_SITES]

static signed char
stencil_sites_y21 [MAX_STENCIL_SITES]

static signed char
stencil_sites_z21 [MAX_STENCIL_SITES]

static signed char
stencil_sites_x25 [MAX_STENCIL_SITES]

static signed char
stencil_sites_y25 [MAX_STENCIL_SITES]

static signed char
stencil_sites_z25 [MAX_STENCIL_SITES]

static signed char
stencil_sites_x27 [MAX_STENCIL_SITES]

static signed char
stencil_sites_y27 [MAX_STENCIL_SITES]

static signed char
stencil_sites_z27 [MAX_STENCIL_SITES]

static signed char
stencil_sites_x33 [MAX_STENCIL_SITES]

static signed char

stencil_sites_y33 [MAX_STENCIL_SITES]

static signed char
stencil_sites_z33 [MAX_STENCIL_SITES]

static signed char
stencil_sites_x57 [MAX_STENCIL_SITES]

static signed char
stencil_sites_y57 [MAX_STENCIL_SITES]

static signed char
stencil_sites_z57 [MAX_STENCIL_SITES]

85.5.1 static unsigned char
initialized *Whether initialized.* 136

85.5.1

static unsigned char **initialized**

Whether initialized.

In file ../Basic/stencil_sites.hh:4542

Whether initialized.

The algorithms to create the stencils are executed just once.

85.1

methods

Names

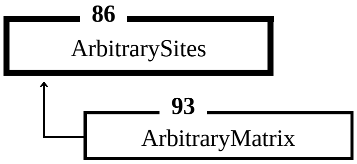
static void **zero_sites** (int sites_size, signed char* sites_x, signed char* sites_y,
signed char* sites_z)
Helper function to reduce code size.

86 class **ArbitrarySites**

An arbitrary group of sites.

In file ../Basic/stencil_sites.hh:4649

Inheritance



Public Members

86.4 **usual methods** 137
86.5 **Data** 141
86.6 **methods** 142

Protected Members

86.1 **copy and assignment helper methods** 142
86.2 **data** 143
86.3 **methods** 143

An arbitrary group of sites.

Author: Alan Louis Scheinine
Version: `$Id: stencil_sites.hh,v 1.5 2002/04/03 22:01:43 alan Exp $`

86.4 usual methods

Names

ArbitrarySites () *Default constructor.*

86.4.1 **ArbitrarySites** (int size_in, const double* stencil_sites_xx)
Constructor. 138

86.4.2 **ArbitrarySites** (int size_in, const double* stencil_sites_xx,
const double* stencil_sites_yy)

Constructor. 139

86.4.3 **ArbitrarySites** (int size_in, const double* stencil_sites_xx,
const double* stencil_sites_yy,
const double* stencil_sites_zz)
Constructor. 139

86.4.4 **ArbitrarySites** (const ReadOnlyNumArray<double>& stencil_sites_xx)
Constructor. 139

86.4.5 **ArbitrarySites** (const ReadOnlyNumArray<double>& stencil_sites_xx,
const ReadOnlyNumArray<double>& stencil_sites_yy)
Constructor. 139

86.4.6 **ArbitrarySites** (const ReadOnlyNumArray<double>& stencil_sites_xx,
const ReadOnlyNumArray<double>& stencil_sites_yy,
const ReadOnlyNumArray<double>& stencil_sites_zz)
Constructor. 140

86.4.7 **ArbitrarySites** (StencilVector_const_ref stencil_sites_xx)
Constructor. 140

86.4.8 **ArbitrarySites** (StencilVector_const_ref stencil_sites_xx,
StencilVector_const_ref stencil_sites_yy)
Constructor. 140

86.4.9 **ArbitrarySites** (StencilVector_const_ref stencil_sites_xx,
StencilVector_const_ref stencil_sites_yy,
StencilVector_const_ref stencil_sites_zz)
Constructor. 140

ArbitrarySites (const ArbitrarySites& object_in)
Copy constructor.

ArbitrarySites&
operator= (const ArbitrarySites& object_in)
Assignment operator.

virtual **˜ArbitrarySites** () *Destructor.*

86.4.1

ArbitrarySites (int size_in, const double* stencil_sites_xx)

Constructor.

In file ../Basic/stencil_sites.hh:4835

Constructor. Sets the x sites of a stencil.

86.4.2

ArbitrarySites (int size_in, const double* stencil_sites_xx, const double* stencil_sites_yy)

Constructor.

In file ../Basic/stencil_sites.hh:4843

Constructor. Sets the x, y sites of a stencil.

86.4.3

ArbitrarySites (int size_in, const double* stencil_sites_xx, const double* stencil_sites_yy, const double* stencil_sites_zz)

Constructor.

In file ../Basic/stencil_sites.hh:4853

Constructor. Sets the x, y, z sites of a stencil.

86.4.4

ArbitrarySites (const ReadOnlyNumArray<double>& stencil_sites_xx)

Constructor.

In file ../Basic/stencil_sites.hh:4865

Constructor. Sets the x sites of a stencil.

86.4.5

ArbitrarySites (const ReadOnlyNumArray<double>& stencil_sites_xx, const ReadOnlyNumArray<double>& stencil_sites_yy)

Constructor.

In file ../Basic/stencil_sites.hh:4871

Constructor. Sets the x, y sites of a stencil.

86.4.6

ArbitrarySites (const ReadOnlyNumArray<double>& stencil_sites_xx, const ReadOnlyNumArray<double>& stencil_sites_yy, const ReadOnlyNumArray<double>& stencil_sites_zz)

Constructor.

In file ../Basic/stencil_sites.hh:4879

Constructor. Sets the x, y, z sites of a stencil.

86.4.7

ArbitrarySites (StencilVector_const_ref stencil_sites_xx)

Constructor.

In file ../Basic/stencil_sites.hh:4889

Constructor. Sets the x sites of a stencil.

86.4.8

ArbitrarySites (StencilVector_const_ref stencil_sites_xx, StencilVector_const_ref stencil_sites_yy)

Constructor.

In file ../Basic/stencil_sites.hh:4895

Constructor. Sets the x, y sites of a stencil.

86.4.9

ArbitrarySites (StencilVector_const_ref stencil_sites_xx, StencilVector_const_ref stencil_sites_yy, StencilVector_const_ref stencil_sites_zz)

Constructor.

In file ../Basic/stencil_sites.hh:4903

Constructor. Sets the x, y, z sites of a stencil.

86.5 Data

Names

86.5.1 ReadOnlyNumArray<double> **stencil_sites_x** *X positions of the sites* 141

86.5.2 ReadOnlyNumArray<double> **stencil_sites_y** *Y positions of the sites* 141

86.5.3 ReadOnlyNumArray<double> **stencil_sites_z** *Z positions of the sites* 141

86.5.1 ReadOnlyNumArray<double> **stencil_sites_x**

X positions of the sites

In file ../Basic/stencil_sites.hh:4939

X positions of the sites

86.5.2 ReadOnlyNumArray<double> **stencil_sites_y**

Y positions of the sites

In file ../Basic/stencil_sites.hh:4942

Y positions of the sites

86.5.3 ReadOnlyNumArray<double> **stencil_sites_z**

Z positions of the sites

In file ../Basic/stencil_sites.hh:4945

Z positions of the sites

86.6

methods

Names

86.6.1	inline	int	**getSize** () const	*Get total size.*	142
86.6.2	inline	int	**getDimension** () const	*Get number of dimensions.*	142

86.6.1

inline int **getSize** () const

Get total size.

In file ../Basic/stencil_sites.hh:4954

Get total size.

Return Value: `number` of sites

86.6.2

inline int **getDimension** () const

Get number of dimensions.

In file ../Basic/stencil_sites.hh:4959

Get number of dimensions.

Return Value: `dimension`

86.1

copy and assignment helper methods

Names

86.1.1	inline	void	**convert** (const ArbitrarySites& object_in)	*Copy of data members*	143
86.1.2	inline	void	**convert_tree** (const ArbitrarySites& object_in)	*Call convert_tree on each parent class then call convert*	143

86.1.1

inline void **convert** (const ArbitrarySites& object_in)

Copy of data members

In file ../Basic/stencil_sites.hh:4658

Copy of data members

86.1.2

inline void **convert_tree** (const ArbitrarySites& object_in)

Call convert_tree on each parent class then call convert

In file ../Basic/stencil_sites.hh:4666

Call convert_tree on each parent class then call convert

86.2 data

Names

int	**_size**	*The size of the array of sites.*
int	**_dimen**	*Number of dimensions.*

86.3 methods

Names

inline void **set_stencil_sites** (int size_in, const double* stencil_sites_xx)

inline void **set_stencil_sites** (int size_in, const double* stencil_sites_xx,
const double* stencil_sites_yy)

inline void **set_stencil_sites** (int size_in, const double* stencil_sites_xx,
const double* stencil_sites_yy,
const double* stencil_sites_zz)

inline void **set_stencil_sites** (const ReadOnlyNumArray<double>&
stencil_sites_xx)

inline void **set_stencil_sites** (const ReadOnlyNumArray<double>& stencil_sites_xx, const ReadOnlyNumArray<double>& stencil_sites_yy)

inline void **set_stencil_sites** (const ReadOnlyNumArray<double>& stencil_sites_xx, const ReadOnlyNumArray<double>& stencil_sites_yy, const ReadOnlyNumArray<double>& stencil_sites_zz)

inline void **set_stencil_sites** (StencilVector_const_ref stencil_sites_xx)

inline void **set_stencil_sites** (StencilVector_const_ref stencil_sites_xx, StencilVector_const_ref stencil_sites_yy)

inline void **set_stencil_sites** (StencilVector_const_ref stencil_sites_xx, StencilVector_const_ref stencil_sites_yy, StencilVector_const_ref stencil_sites_zz)

87 class **StencilSitesTag**

Tag for stencil sites.

In file ../Basic/stencil_params.hh:4988

Public Members

87.2 **methods.** 145

Private Members

87.1 **data** 146

Tag for stencil sites.

The tag that specifies the sites of a particular stencil is wrapped in a class to avoid confusion with the tag that specifies the terms of a polynomial.

Author: Alan Louis Scheinine
Version: `$Id: stencil_params.hh,v 1.5 2002/04/21 01:23:56 alan Exp $`

87.2 methods.

Names

StencilSitesTag () *default constructor*

StencilSitesTag (int tag_in)
constructor

StencilSitesTag (const StencilSitesTag& object_in)
copy constructor

StencilSitesTag&
operator= (const StencilSitesTag& object_in)
assignment

˜StencilSitesTag () *destructor*

87.2.1 inline int **getIntegerValue** () const *Get tag value as an integer.* 146

87.2.1

inline int **getIntegerValue** () const

Get tag value as an integer.

In file ../Basic/stencil_params.hh:5016

Get tag value as an integer.

Return Value: `tag`

87.1

data

Names

int	**_tag**	*Integer-valued tag*

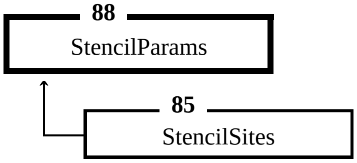
88

class **StencilParams**

Basic parameters for any stencil.

In file ../Basic/stencil_params.hh:5037

Inheritance



Public Members

88.4	**constructors.**	147
88.5	**usual methods.**	148
88.6	**methods.**	148

Protected Members

88.1	**copy and assignment helper methods**	149
88.2	**data**	150
88.3	**methods**	151

Basic parameters for any stencil.

Note, assumes that the stencil tag is equal to the size of the stencil. There is no case in which different stencil patterns have the same size.

Author: Alan Louis Scheinine
Version: `$Id: stencil_params.hh,v 1.5 2002/04/21 01:23:56 alan Exp $`

88.4

constructors.

Names

88.4.1	**StencilParams** ()	*Default constructor.*	148
	StencilParams (int tag_in)		

Defines tag, size, and dimension for a stencil.

StencilParams (const StencilSitesTag& tag_in)
Defines tag, size, and dimension for a stencil.

88.4.1

StencilParams ()

Default constructor.

In file ../Basic/stencil_params.hh:5127

Default constructor. Not really meaningful.

88.5

usual methods.

Names

StencilParams (const StencilParams& object_in)
Copy constructor.

StencilParams&
operator= (const StencilParams& object_in)
Assignment operator.

virtual **˜StencilParams** () *Destructor.*

88.6

methods.

Names

88.6.1	inline	StencilSitesTag	**getTag** () const	*Get tag value.*	149
88.6.2	inline	int	**getSize** () const	*Get total size.*	149
88.6.3	inline	int	**getDimension** () const	*Get number of dimensions (2 or 3).*	149

88.6.1

inline StencilSitesTag **getTag** () const

Get tag value.

In file ../Basic/stencil_params.hh:5162

Get tag value.

Return Value: `tag`

88.6.2

inline int **getSize** () const

Get total size.

In file ../Basic/stencil_params.hh:5167

Get total size.

Return Value: `number` of sites

88.6.3

inline int **getDimension** () const

Get number of dimensions (2 or 3).

In file ../Basic/stencil_params.hh:5172

Get number of dimensions (2 or 3).

Return Value: `dimension`

88.1

copy and assignment helper methods

Names

88.1.1 inline void **convert** (const StencilParams& object_in)
Copy of data members 150

88.1.2 inline void **convert_tree** (const StencilParams& object_in)
Call convert_tree on each parent class then call convert 150

88.1.1

inline void **convert** (const StencilParams& object_in)

Copy of data members

In file ../Basic/stencil_params.hh:5046

Copy of data members

88.1.2

inline void **convert_tree** (const StencilParams& object_in)

Call convert_tree on each parent class then call convert

In file ../Basic/stencil_params.hh:5052

Call convert_tree on each parent class then call convert

88.2

data

Names

StencilSitesTag	**_tag**	*Indicates the stencil choice.*
int	**_size**	*The size of the array of sites.*
int	**_dimen**	*Number of dimensions.*

88.3

methods

Names

	void	**set_stencil_params** (int tag_in)	
88.3.1	bool	**check_stencil_tag** (int tag_in) *check that tag is valid.*	151

88.3.1

bool **check_stencil_tag** (int tag_in)

check that tag is valid.

In file ../Basic/stencil_params.hh:5106

check that tag is valid.

Return Value: `1` if tag is legal, 0 if tag is not a valid choice

Parameters: `tag_in` tag value to test validity

89

class **TermsTag**

Tag for polynomial terms.

In file ../Basic/stencil_terms.hh:5206

Private Members

89.1	**data.**	152
89.2	**methods**	152

Tag for polynomial terms.

The tag that specifies a particular set of terms is wrapped in a class to avoid confusion with the tag that specifies the stencil sites.

Author: Alan Louis Scheinine
Version: `$Id: stencil_terms.hh,v 1.11 2002/04/18 23:38:14 alan Exp $`

89.1

data.

Names

int	**_tag**	*tag for degree of polynomial.*
int	**_basis**	*basis functions.*
int	**_size**	*The size of the array of sites.*
int	**_dimen**	*number of dimensions.*

89.2

methods

Names

void **set_params** ()

89.2.1	**usual methods**	153
89.2.2	**data access methods**	153
89.2.3	**static methods**	154

89.2.1 usual methods

Names

	TermsTag ()	*default constructor*
	TermsTag (int tag_in, int basis_in)	*constructor*
	TermsTag (const TermsTag& object_in)	*copy constructor*
TermsTag&	**operator=** (const TermsTag& object_in)	*assignment*
	˜**TermsTag** ()	*destructor*

89.2.2 data access methods

Names

	inline int	**getIntegerTag** () const			
	inline int	**getIntegerBasis** () const			
89.2.2.1	inline int	**getSize** () const	*Get total size.*	. . . .	153
89.2.2.2	inline int	**getDimension** () const	*Get number of dimensions.*	. . . .	154

89.2.2.1 inline int **getSize** () const

Get total size.

In file ../Basic/stencil_terms.hh:5311

Get total size.

Return Value: `number` of terms

89.2.2.2

inline int **getDimension** () const

Get number of dimensions.

In file ../Basic/stencil_terms.hh:5316

Get number of dimensions.

Return Value: `dimension`

89.2.3

static methods

Names

static bool **check_terms_tag** (int tag_in, int basis_in)

90

class **PrecisionChoice**

Holds choice of stencil and choice of basis for interpolation

In file ../Basic/stencil_terms.hh:5355

Public Members

90.1 **data** 155

Holds choice of stencil and choice of basis for interpolation

90.1

data

Names

int		**stencil_type**
int		**basis_type**
inline	int	**get_stencil_choice** (int dimen) const
inline	int	**get_basis_type** () const

91 class **StencilTerms**

Polynomial terms for a given stencil.

In file ../Basic/stencil_terms.hh:5442

Public Members

91.4	**Constructors**	156
91.5	**Usual methods**	157
91.6	**data**	157
91.7	**methods**	159

Protected Members

91.1	**copy and assignment helper methods**	161
91.2	**data**	162
91.3	**methods**	162

Polynomial terms for a given stencil.

The polynomial terms do not depend on field values. The terms depend on x, y, [and z] values. Each term is a functional such as x*y or x*x*y, etc. with one additional complication that the terms might be averages over a box rather than corresponding to one location.

For a function $f(x) = x^n$, the average value of f in the interval $A < x < B$ is

$$(A^n + A^{n-1} * B + A^{n-2} * B^2 ... A * B^{n-1} + B^n)/(n+1)$$

Author: Alan Louis Scheinine
Version: `$Id: stencil_terms.hh,v 1.11 2002/04/18 23:38:14 alan Exp $`

91.4 Constructors

Names

91.4.1	**StencilTerms** ()	*Default constructor.*	157
	StencilTerms (int tag_in)	*constructor*	
	StencilTerms (int tag_in, int basis_in)	*constructor*	
	StencilTerms (const TermsTag& tag_in)		

constructor

StencilTerms (const TermsTag& tag_in, double x)
constructor, bspline basis functions

StencilTerms (const TermsTag& tag_in, double x, double y)
constructor, bspline basis functions

StencilTerms (const TermsTag& tag_in, double x, double y, double z)
constructor, bspline basis functions

StencilTerms (const TermsTag& tag_in, int dimension_in,
const double* aspect, bool* ok)
constructor, any basis

91.4.1

StencilTerms ()

Default constructor.

In file ../Basic/stencil_terms.hh:5905

Default constructor. Not really meaningful.

91.5

Usual methods

Names

StencilTerms (const StencilTerms& object_in)
Copy constructor.

StencilTerms& **operator=** (const StencilTerms& object_in)
Assignment operator.

virtual **˜StencilTerms** () *Destructor.*

91.6

data

Names

91.6.1 StencilVector **terms** *An array of either the values of the polynomial terms at a location or the values of the polynomial terms with each averaged over a box* 158

91.6.2 StencilVector **xterms** *The x values of an array of vectors for the gradient* 158

91.6.3 StencilVector **yterms** *The y values of an array of vectors for the gradient* 158

91.6.4 StencilVector **zterms** *The z values of an array of vectors for the gradient* 159

91.6.1

StencilVector **terms**

An array of either the values of the polynomial terms at a location or the values of the polynomial terms with each averaged over a box

In file ../Basic/stencil_terms.hh:6146

An array of either the values of the polynomial terms at a location or the values of the polynomial terms with each averaged over a box

91.6.2

StencilVector **xterms**

The x values of an array of vectors for the gradient

In file ../Basic/stencil_terms.hh:6151

The x values of an array of vectors for the gradient

91.6.3

StencilVector **yterms**

The y values of an array of vectors for the gradient

In file ../Basic/stencil_terms.hh:6156

The y values of an array of vectors for the gradient

91.6.4

StencilVector **zterms**

The z values of an array of vectors for the gradient

In file ../Basic/stencil_terms.hh:6161

The z values of an array of vectors for the gradient

91.7

methods

Names

91.7.1 void **make_point_terms** (const double *location)
A function to generate polynomial terms . . . 159

91.7.2 void **make_ntgrl_terms** (const double *box)
A function to generate terms averaged over a box 160

91.7.3 void **make_gradient_terms** (const double *location)
A function to generate gradient vector terms 160

91.7.4 inline TermsTag
getTag () const *Get tag value.* 160

91.7.5 inline int **getSize** () const *Get total size.* 160

91.7.6 inline int **getDimension** () const *Get number of dimensions.* 161

91.7.1

void **make_point_terms** (const double *location)

A function to generate polynomial terms

In file ../Basic/stencil_terms.hh:6172

A function to generate polynomial terms

91.7.2

void **make_ntgrl_terms** (const double *box)

A function to generate terms averaged over a box

In file ../Basic/stencil_terms.hh:6211

A function to generate terms averaged over a box

91.7.3

void **make_gradient_terms** (const double *location)

A function to generate gradient vector terms

In file ../Basic/stencil_terms.hh:6250

A function to generate gradient vector terms

91.7.4

inline TermsTag **getTag** () const

Get tag value.

In file ../Basic/stencil_terms.hh:6290

Get tag value.

Return Value: `tag`

91.7.5

inline int **getSize** () const

Get total size.

In file ../Basic/stencil_terms.hh:6295

Get total size.

Return Value: `number` of terms

91.7.6

inline int **getDimension** () const

Get number of dimensions.

In file ../Basic/stencil_terms.hh:6300

Get number of dimensions.

Return Value: `dimension`

91.1

copy and assignment helper methods

Names

91.1.1 inline void **convert** (const StencilTerms& object_in)
Copy of data members 161

91.1.2 inline void **convert_tree** (const StencilTerms& object_in)
Call convert_tree on each parent class then call convert 161

91.1.1

inline void **convert** (const StencilTerms& object_in)

Copy of data members

In file ../Basic/stencil_terms.hh:5451

Copy of data members

91.1.2

inline void **convert_tree** (const StencilTerms& object_in)

Call convert_tree on each parent class then call convert

In file ../Basic/stencil_terms.hh:5460

Call convert_tree on each parent class then call convert

91.2 data

Names

TermsTag	**_tag**
std::vector<Bspline>	**_basis_functions**

91.3 methods

Names

void	**terms3_taylor** (const double *location)
void	**integrate3_taylor** (const double *box)
void	**gradient3_taylor** (const double *location)
void	**terms5_taylor** (const double *location)
void	**integrate5_taylor** (const double *box)
void	**gradient5_taylor** (const double *location)
void	**terms9_taylor** (const double *location)
void	**integrate9_taylor** (const double *box)
void	**gradient9_taylor** (const double *location)
void	**terms13_taylor** (const double *location)
void	**integrate13_taylor** (const double *box)
void	**gradient13_taylor** (const double *location)
void	**terms27_taylor** (const double *location)
void	**integrate27_taylor** (const double *box)
void	**gradient27_taylor** (const double *location)
void	**terms33_taylor** (const double *location)
void	**integrate33_taylor** (const double *box)
void	**gradient33_taylor** (const double *location)
inline void	**terms_bspline** (const double *location)
inline void	**integrate_bspline** (const double *box)
inline void	**gradient_bspline** (const double *location)
void	**zero_out_stencil_terms** ()
void	**zero_out_terms** ()
void	**zero_out_xyzterms** ()

void	**set_up_terms** ()
void	**set_up_xyzterms** ()
void	**make_basis_functions** (int tag_in, int basis_type, int dimension_in, double x, double y, double z, bool* ok)

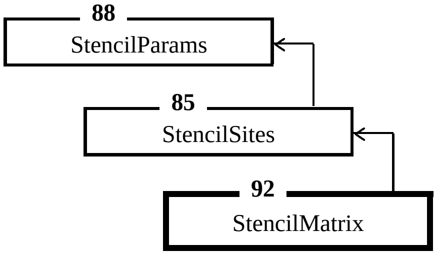
92

class **StencilMatrix** : public StencilSites

A matrix that relates polynomial coefficients to field values.

In file ../Basic/stencil_matrix.hh:6340

Inheritance



Public Members

92.4 **usual methods** 164

92.5 **data** 165

92.6 **methods.** 165

Protected Members

92.1 **copy and assignment helper methods** 166

92.2 **data** 166

92.3 **methods** 167

A matrix that relates polynomial coefficients to field values.

Uses stencils of a regular lattice.

Author: Alan Louis Scheinine
Version: `$Id: stencil_matrix.hh,v 1.7 2002/04/23 23:15:19 alan Exp $`

92.4

usual methods

Names

StencilMatrix () *Default constructor.*

StencilMatrix (int tag_in, int basis_in)
Constructor.

StencilMatrix (const StencilMatrix& object_in)
Copy constructor.

StencilMatrix& **operator=** (const StencilMatrix& object_in)
Assignment.

virtual **˜StencilMatrix** () *Destructor*

92.5 data

Names

92.5.1 vector<double>
_col_by_col_matrix *matrix of polynomial terms and sites.* 165

vector<double>
_row_by_row_matrix

92.5.1 vector<double> **_col_by_col_matrix**

matrix of polynomial terms and sites.

In file ../Basic/stencil_matrix.hh:6510

matrix of polynomial terms and sites. A matrix in which each row contains the terms of a polynomial for a position and the various rows refer to different positions of a stencil.

92.6 methods.

Names

92.6.1 inline int **fill_matrix** (const double *position, const ImageBase *field,
int value_mode)
Fills-in the matrix according to a certain stencil. 166

int **take_inverse** ()

void	**mat_vec_mult** (StencilVector_const_ref stencil_coefs_in, StencilVector_ref stencil_coefs_out) const
void	**vec_mat_mult** (StencilVector_const_ref stencil_coefs_in, StencilVector_ref stencil_coefs_out) const
virtual int	**generateLatticeInverse** (const ImageBase* lattice, int value_mode)
inline int	**numSites** () const
inline int	**num_rows** () const
inline int	**num_cols** () const

92.6.1

inline int **fill_matrix** (const double *position, const ImageBase *field, int value_mode)

Fills-in the matrix according to a certain stencil.

In file ../Basic/stencil_matrix.hh:6524

Fills-in the matrix according to a certain stencil.

Note, assumes that the stencil tag is equal to the size of the stencil. There is no case in which different stencil patterns have the same size.

92.1

copy and assignment helper methods

Names

inline void	**convert** (const StencilMatrix& object_in)	*Copy of data members.*
inline void	**convert_tree** (const StencilMatrix& object_in)	*Call convert_tree on each parent class then call convert.*

92.2

data

Names

TermsTag	**_terms_tag**
int	**_num_rows**
int	**_num_cols**

92.3 methods

Names

92.3.1 inline int **fill_matrix** (int syze, const double* position, const double* aspect, int value_mode)
Fill matrix based on field for a specific stencil. 167

int **fill_matrix_n** (int syze, int dymen, StencilTerms& stencil, const double* position, const double* aspect, int value_mode)
Core of filling procedure.

92.3.1 inline int **fill_matrix** (int syze, const double* position, const double* aspect, int value_mode)

Fill matrix based on field for a specific stencil.

In file ../Basic/stencil_matrix.hh:6389

Fill matrix based on field for a specific stencil.

Parameters:		
	`positionCan`	use a different interpolation function aroundeach pixel/voxel,hence, the position can be zero (0.0,0.0[,0.0]).
	`field`	image values
	`value_modePOINT_VALUE_MODE`	impliesmatrix terms are the value at the center point.BOX_VALUE_MODE impliesmatrix terms are an average over the pixel/voxel.

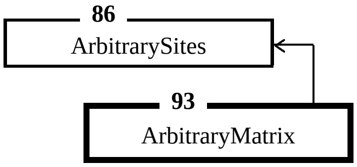
93

class **ArbitraryMatrix** : public ArbitrarySites

A matrix that relates polynomial coefficients to field values.

In file ../Basic/stencil_matrix.hh:6586

Inheritance



Public Members

93.4 **usual methods** 168
93.5 **data** 170
93.6 **methods.** 170

Protected Members

93.1 **copy and assignment helper methods** 171
93.2 **data** 172
93.3 **methods** 172

A matrix that relates polynomial coefficients to field values.

Uses specific polynomial coefficients defined in class StencilTerms but has arbitrary sites.

Author: Alan Louis Scheinine
Version: `$Id: stencil_matrix.hh,v 1.7 2002/04/23 23:15:19 alan Exp $`

93.4

usual methods

Names

ArbitraryMatrix () *Default constructor.*

ArbitraryMatrix (ArbitrarySites& object_in, int tag_in, int basis_in)

Constructor.

ArbitraryMatrix (int num_sites_in, const double* stencil_sites_xx, int tag_in, int basis_in)
Constructor.

ArbitraryMatrix (int num_sites_in, const double* stencil_sites_xx, const double* stencil_sites_yy, int tag_in, int basis_in)
Constructor.

ArbitraryMatrix (int num_sites_in, const double* stencil_sites_xx, const double* stencil_sites_yy, const double* stencil_sites_zz, int tag_in, int basis_in)
Constructor.

ArbitraryMatrix (const ReadOnlyNumArray<double>& stencil_sites_xx, int tag_in, int basis_in)
Constructor.

ArbitraryMatrix (const ReadOnlyNumArray<double>& stencil_sites_xx, const ReadOnlyNumArray<double>& stencil_sites_yy, int tag_in, int basis_in)
Constructor.

ArbitraryMatrix (const ReadOnlyNumArray<double>& stencil_sites_xx, const ReadOnlyNumArray<double>& stencil_sites_yy, const ReadOnlyNumArray<double>& stencil_sites_zz, int tag_in, int basis_in)
Constructor.

ArbitraryMatrix (StencilVector_const_ref stencil_sites_xx, int tag_in, int basis_in)
Constructor.

ArbitraryMatrix (StencilVector_const_ref stencil_sites_xx, StencilVector_const_ref stencil_sites_yy, int tag_in, int basis_in)
Constructor.

ArbitraryMatrix (StencilVector_const_ref stencil_sites_xx, StencilVector_const_ref stencil_sites_yy, StencilVector_const_ref stencil_sites_zz, int tag_in, int basis_in)
Constructor.

ArbitraryMatrix (const ArbitraryMatrix& object_in)
Copy constructor.

ArbitraryMatrix&
operator= (const ArbitraryMatrix& object_in)
Assignment.

virtual ~**ArbitraryMatrix** () *Destructor*

93.5 data

Names

93.5.1	vector<double>	**_col_by_col_matrix**	*matrix of polynomial terms and sites.*	170
	vector<double>	**_row_by_row_matrix**		

93.5.1 vector<double> **_col_by_col_matrix**

matrix of polynomial terms and sites.

In file ../Basic/stencil_matrix.hh:6808

matrix of polynomial terms and sites. A matrix in which each row contains the terms of a polynomial for a position and the various rows refer to different positions of a stencil.

93.6 methods.

Names

93.6.1	inline TermsTag	**getTag** () const	*Get tag value.*	171
93.6.2	inline int	**numTerms** () const	*Get number of terms.*	171
93.6.3	inline int	**numSites** () const	*Get number of sites.*	171
	inline int	**num_rows** () const		
	inline int	**num_cols** () const		
	void	**fill_matrix** ()		
	StencilMatrix	**makeInnerSquared** () const		
	void	**makeInnerSquared** (StencilMatrix& tmp) const		

93.6.1

inline TermsTag **getTag** () const

Get tag value.

In file ../Basic/stencil_matrix.hh:6819

Get tag value.

Return Value: `tag`

93.6.2

inline int **numTerms** () const

Get number of terms.

In file ../Basic/stencil_matrix.hh:6824

Get number of terms.

Return Value: `number` of terms

93.6.3

inline int **numSites** () const

Get number of sites.

In file ../Basic/stencil_matrix.hh:6829

Get number of sites.

Return Value: `number` of sites

93.1

copy and assignment helper methods

Names

inline void **convert** (const ArbitraryMatrix& object_in)
Copy of data members.

inline void **convert_tree** (const ArbitraryMatrix& object_in)
Call convert_tree on each parent class then call convert.

93.2 data

Names

int	**_num_sites**	*Number of sites.*
TermsTag	**_terms_tag**	*Indicates the polynomial terms choice.*
int	**_num_rows**	
int	**_num_cols**	

93.3 methods

Names

void **check_consistency** ()

94

typedef TNTVect<double> **StencilVector**

An array of double precision numbers.

In file ../Basic/stencil_vector.hh:6904

An array of double precision numbers.

Note, unlike an STL vector, resizing destroys the contents.

Author: Alan Louis Scheinine
Version: `$Id: stencil_vector.hh,v 1.3 2002/04/10 21:00:43 alan Exp $`

95

typedef TNTVect<double> * **StencilVector_pointer**

In file ../Basic/stencil_vector.hh:6906

96

typedef TNTVect<double> & **StencilVector_ref**

In file ../Basic/stencil_vector.hh:6908

97

typedef const TNTVect<double> * **StencilVector_const_pointer**

In file ../Basic/stencil_vector.hh:6910

98

typedef const TNTVect<double> & **StencilVector_const_ref**

In file ../Basic/stencil_vector.hh:6912

99

typedef TNT::Vector<double> iterator **StencilVector_iterator**

In file ../Basic/stencil_vector.hh:6914

100

typedef TNT::Vector<double> const_iterator **StencilVector_const_iterator**

In file ../Basic/stencil_vector.hh:6916

101

StencilVector_pointer **newStencilVector** ()

In file ../Basic/stencil_vector.hh:6918

102

StencilVector_pointer **newStencilVector** (int n)

In file ../Basic/stencil_vector.hh:6920

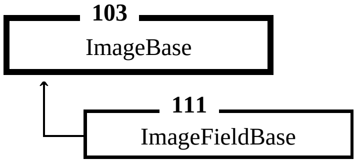
103

class **ImageBase**

Contains information that describes a field.

In file ../Basic/image_base.hh:6957

Inheritance



Public Members

103.4 **static constants** 183

103.5 **usual methods** 183

103.6 **virtual methods** 183

103.7 **methods** 185

Protected Members

103.1 **copy and assignment helper methods** 185

103.2 **data** 186

103.3 **methods** 186

Contains information that describes a field.

A few words should be said about **_aspect** and **inverse_aspect**. The terminology “aspect” is used because for non-linear interpolation the aspect ratio is important. But in addition, it is recommended that the **_aspect** array be considered a constant that indicates the physical size because in this way two different fields can be compared.

The array **inverse_aspect** is a duplication of information, it is simply the inverse of the **_aspect** array. It is generated because the inverse values are used often in a particular calculation.

Author: Alan Louis Scheinine
Version: `$Id: image_base.hh,v 1.2 2002/03/16 17:09:31 alan Exp $`

103.4 static constants

Names

static const int **ImageBase::IMAGE_DATA_TYPE_NONE**

static const int **ImageBase::IMAGE_DATA_TYPE_CHAR**

static const int **ImageBase::IMAGE_DATA_TYPE_SIGNED_CHAR**

static const int **ImageBase::IMAGE_DATA_TYPE_UNSIGNED_CHAR**

static const int **ImageBase::IMAGE_DATA_TYPE_SHORT**

static const int **ImageBase::IMAGE_DATA_TYPE_UNSIGNED_SHORT**

static const int **ImageBase::IMAGE_DATA_TYPE_INT**

static const int **ImageBase::IMAGE_DATA_TYPE_UNSIGNED_INT**

static const int **ImageBase::IMAGE_DATA_TYPE_LONG**

static const int **ImageBase::IMAGE_DATA_TYPE_UNSIGNED_LONG**

static const int **ImageBase::IMAGE_DATA_TYPE_FLOAT**

static const int **ImageBase::IMAGE_DATA_TYPE_DOUBLE**

static const int **ImageBase::IMAGE_DATA_TYPE_LONG_DOUBLE**

103.5 usual methods

Names

ImageBase () *default constructor*

ImageBase (int dimension, const int *lattice_bounds, const double *aspect_ratio)
constructor

ImageBase (const ImageBase& object_in)
copy constructor

ImageBase& **operator=** (const ImageBase& object_in)
assignment operator

virtual **~ImageBase** () *destructor*

103.6 virtual methods

Names

virtual void **set_Bounds** (const int *lattice_bounds)

virtual int **get_Bounds** (int i) const

virtual void **set_Aspect** (const double *aspect_ratio)

virtual double **get_Aspect** (int i) const

virtual void **set_Dimension** (unsigned char dimension)

virtual unsigned char
get_Dimension () const

103.6.1 virtual bool **find_Indices_Nearest** (const double* const coord,
int* const lattice_site,
double* const location) const
Finds nearest lattice pixel or voxel and displacement. 184

103.6.2 virtual bool **find_Indices_Nearest** (const double* const coord,
int* const lattice_site, double* const location,
const signed char* field_pos) const
Finds nearest lattice pixel or voxel and displacement. 185

103.6.1

virtual bool **find_Indices_Nearest** (const double* const coord, int* const lattice_site, double* const location) const

Finds nearest lattice pixel or voxel and displacement.

In file ../Basic/image_base.hh:7130

Finds nearest lattice pixel or voxel and displacement.

Return Value: `true` if point is inside the lattice

Parameters:
`coord` (input) coordinate of a point
`lattice_site` (output) point is inside this pixel or voxel
`location` (output) displacement from middle of pixel or voxel

103.6.2

virtual bool **find_Indices_Nearest** (const double* const coord, int* const lattice_site, double* const location, const signed char* field_pos) const

Finds nearest lattice pixel or voxel and displacement.

In file ../Basic/image_base.hh:7141

Finds nearest lattice pixel or voxel and displacement.

Return Value:	`true`	if point is inside the lattice
Parameters:	`coord`	(input) coordinate of a point
	`lattice_site`	(output) point is inside this pixel or voxel
	`location`	(output) displacement from middle of pixel or voxel
	`field_pos`	(input) coord zero is lower edge, middle, or upper edge

103.7 methods

Names

const int*	**getBoundsArray** () const
const double*	**getAspectArray** () const

103.1 copy and assignment helper methods

Names

103.1.1 void **convert** (const ImageBase& object_in)
Copy of data members 185

103.1.2 void **convert_tree** (const ImageBase& object_in)
Call convert_tree on each parent class then call convert 186

103.1.1 void **convert** (const ImageBase& object_in)

Copy of data members

In file ../Basic/image_base.hh:6968

Copy of data members

103.1.2

void **convert_tree** (const ImageBase& object_in)

Call convert_tree on each parent class then call convert

In file ../Basic/image_base.hh:6978

Call convert_tree on each parent class then call convert

103.2

data

Names

unsigned char	**_dimen**	*number of dimensions*
int	**_bounds** [3]	*size of the array image in each direction*
double	**_aspect** [3]	*real-valued size of the pixel or voxel*
double	**inverse_aspect** [3]	*inverses of the values of _aspect*

103.3

methods

Names

void	**init_image_base** ()	*sets default values for data*
bool	**check_dimension** (int dimension)	*checks that the number of dimensions is between 1 and 3*

104

namespace **BasicDataType**

conversion from a type or character array to an integer descriptor

In file ../Basic/image_base.hh:0

Names

104.1	template<class U> int	**toDataType** ()	*Conversion from a type an integer descriptor.*	187
104.2	int	**toDataType** (const char* type_in)	*Conversion from a character array to an integer descriptor.*	187

conversion from a type or character array to an integer descriptor

104.1

template<class U> int **toDataType** ()

Conversion from a type an integer descriptor.

In file ../Basic/image_base.hh:7179

Conversion from a type an integer descriptor.

Return Value: `integer` descriptor

104.2

int **toDataType** (const char* type_in)

Conversion from a character array to an integer descriptor.

In file ../Basic/image_base.hh:7198

Conversion from a character array to an integer descriptor.

Return Value: `integer` descriptor

105

template<class T> class **ObjVar**

ObjVar takes control of a pointer.

In file ../Field/obj_var.hh:7235

Public Members

105.3	**methods**	188

Private Members

105.1	**Methods.**	189
105.2	**data**	189

ObjVar takes control of a pointer.

A counted pointer that takes control of a pointer.

Modified from omniORB2 objectAdapter.h Created on: 5/3/99 Author : David Riddoch (djr) Copyright (C) 1996, 1999 AT&T Research Cambridge This file is part of the omniORB library. The omniORB library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

The counted pointer with a counter from the heap is shown in a Stroustrup book on C++.

Author: Alan Louis Scheinine
Version: `$Id: obj_var.hh,v 1.1 2002/04/03 22:01:43 alan Exp $`

105.3

methods

Names

	ObjVar ()	*default constructor*
	ObjVar (T* p)	*Constructor, now the ObjVar is responsible for deleting T* p.*
	ObjVar (const ObjVar<T>& r)	*copy constructor*
	˜ObjVar ()	*destructor*
ObjVar<T> &	**operator=** (T* p)	*Assignment, now the ObjVar is responsible for deleting T* p.*
ObjVar<T> &	**operator=** (const ObjVar<T>& r)	

		assignment
inline T*	**operator->** () const	
inline	**operator T*** () const	
inline	**operator T*&** ()	

105.1 Methods.

Names

void	**release_ptr** ()	*reduce the reference count and maybe delete pointer and counter*
void	**duplicate_ptr** (const ObjVar<T>& r)	*Become another carrier of the pointer.*

105.2 data

Names

T*	**ptr_**
int*	**reference_count_**

106

class **LinTrans**

A linear transformation.

In file ../Field/lin_trans.hh:7332

Public Members

106.1	**data**	190
106.3	**usual methods**	190
106.4	**methods**	191

Protected Members

106.2	**copy and assignment helper methods**	191

A linear transformation.

A 3 by 3 linear transform.

Author: Alan Louis Scheinine
Version: `$Id: lin_trans.hh,v 1.1.1.1 2002/02/22 23:22:30 alan Exp $`

106.1

data

Names

double	**_mat** [9]	*Three by three matrix for rotations and scaling.*
double	**_vec** [3]	*Three components of translation.*

106.3

usual methods

Names

	LinTrans (const LinTrans& object_in)
LinTrans&	**operator=** (const LinTrans& object_in)
virtual	**~LinTrans** ()

106.4

methods

Names

void	**transform_identity** ()
void	**transform_translate** (double x, double y, double z)
void	**pre_transform_translate** (double x, double y, double z)
void	**post_transform_translate** (double x, double y, double z)
void	**transform_scale** (double x, double y, double z)
void	**pre_transform_scale** (double x, double y, double z)
void	**post_transform_scale** (double x, double y, double z)
void	**transform_rotate** (double degrees, double x_axis, double y_axis, double z_axis)
void	**pre_transform_rotate** (double degrees, double x_axis, double y_axis, double z_axis)
void	**post_transform_rotate** (double degrees, double x_axis, double y_axis, double z_axis)

106.2

copy and assignment helper methods

Names

void	**convert** (const LinTrans& object_in) *Copy of data members.*
void	**convert_tree** (const LinTrans& object_in) *Call convert_tree on each parent class then call convert.*

107

class **MapDef**

Linear mapping between two fields.

In file ../Field/map_def.hh:7493

Public Members

107.1	**data**	193
107.3	**usual methods**	194
107.4	**methods**	194

Protected Members

107.2	**copy and assignment helper methods**	194

Linear mapping between two fields.

The class MapDef has a public data member LinTrans linear that is applied when map_a_point is called.

The value of beg_field_pos and end_field_pos can be used to move the coordinate system in a way that has a simple meaning without the need to specify a precise value for the linear transformation. The variable beg_field_pos is an array for which the first element refers to the x direction, the second element refers to the y direction, and beg_field_pos[2] refers to the z direction. The same applies to the array end_field_pos. A value of -1 for first element for the domain of the map, (i.e. beg_field_pos[0] == -1) means that the rotation matrix will operate with axis at the center of the field if the x coordinate value of zero is used for an edge of the lattice. That is, if the user gives a value of 0.0 for the x position of a point and the user intends that 0.0 refers to the edge of the field (typically a corner since x, y, and z typically follow the same scheme) then the rotation matrix of the linear transformation will rotate the field about the center. In other words, the value given as the starting point as one half the field width subtracted from it before the rotation is applied. A value of 0 means that the coordinate system is centered in the corresponding direction. A value of 1 means to assign as position 0.0 the side that would have the highest index value of the field array, an unlikely coordinate scheme. Likewise, the array end_field_pos controls the interpretation of the position values for the destination. A further example, with regard to the use of the function map_a_point(), setting all values of the arrays beg_field_pos and end_field_pos to zero gives a rotation about the center of the two fields of the map when the center is the zero of the coordinate system.

The advantage of using beg_field_pos and end_field_pos is that the user does not need to find the size of the fields and then change the member linear.vec[3] to achieve any of these simple translations.

Note that the function map_a_point() uses positions in ”real” space where the voxel size is the ”real” size of a voxel. In other words, a point near the 100th lattice position in the x direction does not imply a position near 100.0. Instead, the point would have an x value near (for example) 300.0 if the voxel size was 3.0 and if the coordinate system had (0, 0, 0) at one corner.

Author: Alan Louis Scheinine
Version: `$Id: map_def.hh,v 1.2 2002/03/08 03:19:56 alan Exp $`

107.1 data

Names

107.1.1	LinTrans	**linear**	*a linear transformation*	193
107.1.2	signed char	**beg_field_pos** [3]	*left justify, center, or right justify initial field*	193
107.1.3	signed char	**end_field_pos** [3]	*left justify, center, or right justify destination field*	194

107.1.1 LinTrans **linear**

a linear transformation

In file ../Field/map_def.hh:7505

a linear transformation

A linear transformation given by a matrix and a vector translation.

107.1.2 signed char **beg_field_pos** [3]

left justify, center, or right justify initial field

In file ../Field/map_def.hh:7513

left justify, center, or right justify initial field

Left justify, center, or right justify relative to the initial field (possible values of -1, 0, or 1) for each of three directions.

107.1.3 signed char **end_field_pos** [3]

left justify, center, or right justify destination field

In file ../Field/map_def.hh:7521

left justify, center, or right justify destination field

Left justify, center, or right justify relative to the destination field (possible values of -1, 0, or 1) for each of three directions.

107.3

usual methods

Names

	MapDef ()	*default constructor*
	MapDef (const MapDef& object_in)	*copy constructor*
MapDef&	**operator=** (const MapDef& object_in)	*assignment*
virtual	**˜MapDef** ()	*destructor*

107.4

methods

Names

int	**map_a_point** (const ImageBase& field_in, const ImageBase& field_out, const double *location_in, double *location_out) const	

107.2

copy and assignment helper methods

Names

inline void	**convert** (const MapDef& object_in)	*Copy of data members.*
inline void	**convert_tree** (const MapDef& object_in)	*Call convert_tree on each parent class then call convert.*

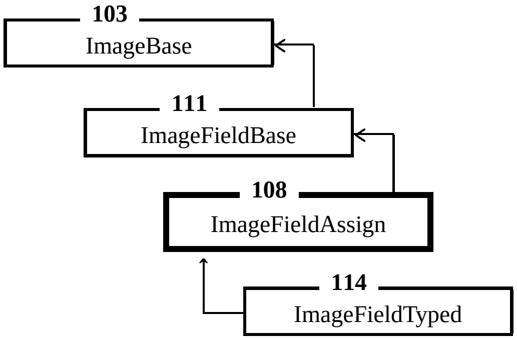
108

template<class T> class **ImageFieldAssign** : public ImageFieldBase<T>

Allows copying and assignment between different types.

In file ../Field/image_field_assign.hh:7643

Inheritance



Public Members

108.1 **type definitions** 196
108.2 **access** 196
108.5 **usual methods** 197
108.6 **access** 197
108.7 **arithmetic methods** 197
108.8 **static methods** 201
108.9 **copy and assignment between different types** 201
108.10 **I/O** 201

Protected Members

108.3 **helper methods for copy and assignment between types** 202
108.4 **copy and assignment helper methods** 202

Allows copying and assignment between different types.

The public methods of TNT vector are redefined here. ImageFieldBase<T> has the TNT vector but defining the methods in that class means too many levels of redefinitions.

For the following operators

```
template<class U>
ImageFieldAssign<T>& operator+=(const ImageFieldAssign<U> &A)
```

```
template<class U>
ImageFieldAssign<T>& operator-=(const ImageFieldAssign<U> &A)
template<class U>
ImageFieldAssign<T>& operator*=(const ImageFieldAssign<U> &A)
template<class U>
ImageFieldAssign<T>& operator/=(const ImageFieldAssign<U> &A)
```

if A has a length, width or height greater that the instantation on the left hand side, then some of the data of A is lost. If A is smaller than the left hand side in a particular direction, part of the left hand side is not changed. The left hand and right hand sides of the operator are aligned along the sides that correspond to the zero indices. In general, these are simple arithmetic operators and should be used for fields of equal shape and size. The reason for defining the operators for fields of unequal shapes is so that the more generalized usage will not crash the program.

Author: Alan Louis Scheinine
Version: `$Id: image_field_assign.hh,v 1.5 2002/04/10 21:00:43 alan Exp $`

108.1 type definitions

Names

typedef Subscript **size_type**

typedef T **value_type**

typedef T **element_type**

typedef T* **pointer**

typedef T* **iterator**

typedef T& **reference**

typedef const T* **const_iterator**

typedef const T& **const_reference**

108.2 access

Names

inline T* **begin** ()

inline T* **end** ()

inline const T*
begin () const

inline const T*
end () const

108.5 usual methods

Names

ImageFieldAssign () *default constructor*

ImageFieldAssign (int dimension, const int *lattice_bounds, const double *aspect_ratio, const T& value = T(0))
constructor

ImageFieldAssign (const ImageFieldAssign<T>& object_in)
copy constructor

ImageFieldAssign<T> &
operator= (const ImageFieldAssign<T>& object_in)
assignment

virtual **˜ImageFieldAssign** ()

108.6 access

Names

inline **operator const TNTVect<T>&** () const

inline **operator TNTVect<T>&** ()

inline T& **operator[]** (Subscript i)

inline const T&
operator[] (Subscript i) const

108.7 arithmetic methods

Names

inline ImageFieldAssign<T> &

operator= (const Number& scalar)

inline ImageFieldAssign<T> &
operator+= (const Number& scalar)

inline ImageFieldAssign<T> &
operator-= (const Number& scalar)

inline ImageFieldAssign<T> &
operator*= (const Number& scalar)

inline ImageFieldAssign<T> &
operator/= (const Number& scalar)

inline ImageFieldAssign<T> &
operator= (char scalar)

inline ImageFieldAssign<T> &
operator+= (char scalar)

inline ImageFieldAssign<T> &
operator-= (char scalar)

inline ImageFieldAssign<T> &
operator*= (char scalar)

inline ImageFieldAssign<T> &
operator/= (char scalar)

inline ImageFieldAssign<T> &
operator= (signed char scalar)

inline ImageFieldAssign<T> &
operator+= (signed char scalar)

inline ImageFieldAssign<T> &
operator-= (signed char scalar)

inline ImageFieldAssign<T> &
operator*= (signed char scalar)

inline ImageFieldAssign<T> &
operator/= (signed char scalar)

inline ImageFieldAssign<T> &
operator= (unsigned char scalar)

inline ImageFieldAssign<T> &
operator+= (unsigned char scalar)

inline ImageFieldAssign<T> &
operator-= (unsigned char scalar)

inline ImageFieldAssign<T> &
operator*= (unsigned char scalar)

inline ImageFieldAssign<T> &
operator/= (unsigned char scalar)

inline ImageFieldAssign<T> &
operator= (short scalar)

inline ImageFieldAssign<T> &
operator+= (short scalar)

inline ImageFieldAssign<T> &

operator-= (short scalar)

inline ImageFieldAssign<T> &
operator*= (short scalar)

inline ImageFieldAssign<T> &
operator/= (short scalar)

inline ImageFieldAssign<T> &
operator= (unsigned short scalar)

inline ImageFieldAssign<T> &
operator+= (unsigned short scalar)

inline ImageFieldAssign<T> &
operator-= (unsigned short scalar)

inline ImageFieldAssign<T> &
operator*= (unsigned short scalar)

inline ImageFieldAssign<T> &
operator/= (unsigned short scalar)

inline ImageFieldAssign<T> &
operator= (int scalar)

inline ImageFieldAssign<T> &
operator+= (int scalar)

inline ImageFieldAssign<T> &
operator-= (int scalar)

inline ImageFieldAssign<T> &
operator*= (int scalar)

inline ImageFieldAssign<T> &
operator/= (int scalar)

inline ImageFieldAssign<T> &
operator= (unsigned int scalar)

inline ImageFieldAssign<T> &
operator+= (unsigned int scalar)

inline ImageFieldAssign<T> &
operator-= (unsigned int scalar)

inline ImageFieldAssign<T> &
operator*= (unsigned int scalar)

inline ImageFieldAssign<T> &
operator/= (unsigned int scalar)

inline ImageFieldAssign<T> &
operator= (long scalar)

inline ImageFieldAssign<T> &
operator+= (long scalar)

inline ImageFieldAssign<T> &
operator-= (long scalar)

inline ImageFieldAssign<T> &
operator*= (long scalar)

inline ImageFieldAssign<T> &

operator/= (long scalar)

inline ImageFieldAssign<T> &
operator= (unsigned long scalar)

inline ImageFieldAssign<T> &
operator+= (unsigned long scalar)

inline ImageFieldAssign<T> &
operator-= (unsigned long scalar)

inline ImageFieldAssign<T> &
operator*= (unsigned long scalar)

inline ImageFieldAssign<T> &
operator/= (unsigned long scalar)

inline ImageFieldAssign<T> &
operator= (float scalar)

inline ImageFieldAssign<T> &
operator+= (float scalar)

inline ImageFieldAssign<T> &
operator-= (float scalar)

inline ImageFieldAssign<T> &
operator*= (float scalar)

inline ImageFieldAssign<T> &
operator/= (float scalar)

inline ImageFieldAssign<T> &
operator= (double scalar)

inline ImageFieldAssign<T> &
operator+= (double scalar)

inline ImageFieldAssign<T> &
operator-= (double scalar)

inline ImageFieldAssign<T> &
operator*= (double scalar)

inline ImageFieldAssign<T> &
operator/= (double scalar)

inline ImageFieldAssign<T> &
operator= (long double scalar)

inline ImageFieldAssign<T> &
operator+= (long double scalar)

inline ImageFieldAssign<T> &
operator-= (long double scalar)

inline ImageFieldAssign<T> &
operator*= (long double scalar)

inline ImageFieldAssign<T> &
operator/= (long double scalar)

template<class U>inline ImageFieldAssign<T> &
operator+= (const ImageFieldAssign<U> &A)

template<class U>inline ImageFieldAssign<T> &

operator-= (const ImageFieldAssign<U> &A)

template<class U>inline ImageFieldAssign<T> &
operator*= (const ImageFieldAssign<U> &A)

template<class U>inline ImageFieldAssign<T> &
operator/= (const ImageFieldAssign<U> &A)

108.8

static methods

Names

static int **getStaticDataType** ()

108.9

copy and assignment between different types

Names

template<class U>
ImageFieldAssign (const ImageFieldBase<U>& object_in)

template<class U> ImageFieldAssign<T> &
operator= (const ImageFieldBase<U>& object_in)

108.10

I/O

Names

inline std::ostream&
put (std::ostream& s) const

inline std::istream&
get (std::istream& s)

108.3

helper methods for copy and assignment between types

Names

template<class U> void
assign (const ImageFieldBase<U>& object_in)

template<class U> void
convert_tree_base (const ImageFieldBase<U>& object_in)

108.4

copy and assignment helper methods

Names

inline	void	**convert** (const ImageFieldAssign<T>& object_in)
inline	void	**convert_tree** (const ImageFieldAssign<T>& object_in)

109

template<class T> std::ostream& **operator<<** (std::ostream &s, const ImageFieldAssign<T> &A)

ImageFieldAssign write to standard output.

In file ../Field/image_field_assign.hh:8354

ImageFieldAssign write to standard output.

110

template<class T> std::istream& **operator>>** (std::istream &s, ImageFieldAssign<T> &A)

ImageFieldAssign read from standard input.

In file ../Field/image_field_assign.hh:8358

ImageFieldAssign read from standard input.

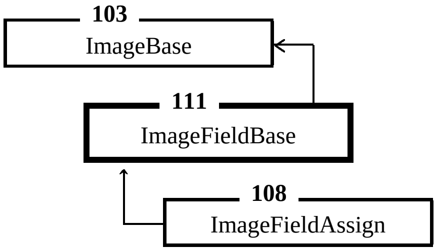
111

template<class T> class **ImageFieldBase** : public ImageBase

Contains an array of field values.

In file ../Field/image_field_base.hh:8380

Inheritance



Public Members

111.3 **usual methods** 205
111.4 **redefine some virtual functions of ImageBase** 206
111.5 **virtual methods** 206
111.6 **methods** 206

Protected Members

111.1 **data** 207
111.2 **copy and assignment helper methods** 207

Contains an array of field values.

The type given in the template specifies the type of the array that represents the field.

Author: Alan Louis Scheinine
Version: `$Id: image_field_base.hh,v 1.5 2002/04/10 21:00:43 alan Exp $`

111.3

usual methods

Names

ImageFieldBase () *default constructor*

ImageFieldBase (int dimension, const int *lattice_bounds,
const double *aspect_ratio, const T& value = T(0))
constructor

ImageFieldBase (const ImageFieldBase<T>& object_in)
copy constructor

ImageFieldBase<T> &
operator= (const ImageFieldBase<T>& object_in)
assignment

ImageFieldBase<T> &
operator= (const T& scalar)
assignment

virtual **˜ImageFieldBase** () *destructor*

111.4

redefine some virtual functions of ImageBase

Names

void **set_Dimension** (unsigned char dimension)
cannot change the number of dimensions

void **set_Bounds** (const int *lattice_bounds)
cannot change the size of the lattice

111.5

virtual methods

Names

virtual int **get_Data_Type** () const

111.6

methods

Names

inline const TNTVect<T> &

getImageArray () const

inline TNTVect<T> & **getImageArray** ()

111.1 data

Names

TNTVect<T>	**image**	*the field*

111.2 copy and assignment helper methods

Names

inline void **convert** (const ImageFieldBase<T>& object_in)

inline void **convert_tree** (const ImageFieldBase<T>& object_in)

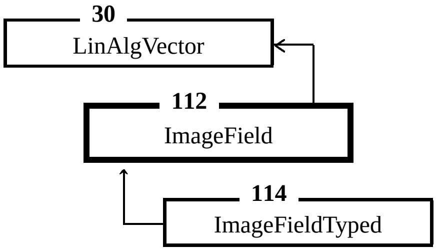
112

class **ImageField** : virtual public LinAlgVector

Abstract class that declares image methods.

In file ../Field/image_field.hh:8534

Inheritance



Public Members

112.1 **constructors to pass information to LinAlgVector** 208
112.2 **virtual functions** 209

Abstract class that declares image methods.

Declares (as virtual functions) the basic methods for data of an image or a field.

Author: Alan Louis Scheinine
Version: `$Id: image_field.hh,v 1.8 2002/04/18 23:38:14 alan Exp $`

112.1

constructors to pass information to LinAlgVector

Names

ImageField () *default constructor*

ImageField (const LinAlgVector& lav)
constructor

112.2 virtual functions

Names

virtual **~ImageField** ()

virtual void **setBounds** (const int *lattice_bounds)

virtual int **getBounds** (int i) const

virtual void **setAspect** (const double *aspect_ratio)

virtual double **getAspect** (int i) const

virtual void **setDimension** (unsigned char dimension)

virtual unsigned char
getDimension () const

virtual const int* const
getBounds_array () const

virtual const double* const
getAspect_array () const

virtual bool **find_indices_nearest** (const double* const coord, int* const lattice_site,
double* const location) const

virtual bool **find_indices_nearest** (const double* const coord, int* const lattice_site,
double* const location,
const signed char* field_pos) const

virtual const ImageBase*
getImageBase () const

virtual ImageField*
new_extend_by_two () const

virtual int **make_stencil_rhs** (StencilVector_ref rhs, const StencilSites& sites,
const int* lattice_site, bool check_bounds) const

virtual int **check_template_with_field** (int dimension_must_be,
const int* lattice_site, int extent) const

112.2.1 virtual ImageField*
new_by_interpol (const ImageBase& grid, const MapDef& mapping,
int precision_level, const PrecisionChoice& pc,
int debug) const
generate field by interpolation 213

virtual int **getDataType** () const

virtual int **toDataType** (const char* type_in)

virtual ImageField*
newImageField () const

virtual ImageField*

newImageField (int dimension, const int *lattice_bounds,
const double *aspect_ratio) const

virtual ImageField*
cloneImageField () const

virtual Number
getImage_Number (int ix) const

virtual Number
getImage_Number (int ix, int iy) const

virtual Number
getImage_Number (int ix, int iy, int iz) const

virtual char **getImage_char** (int ix) const

virtual char **getImage_char** (int ix, int iy) const

virtual char **getImage_char** (int ix, int iy, int iz) const

virtual signed char
getImage_signed_char (int ix) const

virtual signed char
getImage_signed_char (int ix, int iy) const

virtual signed char
getImage_signed_char (int ix, int iy, int iz) const

virtual unsigned char
getImage_unsigned_char (int ix) const

virtual unsigned char
getImage_unsigned_char (int ix, int iy) const

virtual unsigned char
getImage_unsigned_char (int ix, int iy, int iz) const

virtual short **getImage_short** (int ix) const

virtual short **getImage_short** (int ix, int iy) const

virtual short **getImage_short** (int ix, int iy, int iz) const

virtual unsigned short
getImage_unsigned_short (int ix) const

virtual unsigned short
getImage_unsigned_short (int ix, int iy) const

virtual unsigned short
getImage_unsigned_short (int ix, int iy, int iz) const

virtual int **getImage_int** (int ix) const

virtual int **getImage_int** (int ix, int iy) const

virtual int **getImage_int** (int ix, int iy, int iz) const

virtual unsigned int
getImage_unsigned_int (int ix) const

virtual unsigned int
getImage_unsigned_int (int ix, int iy) const

virtual unsigned int

getImage_unsigned_int (int ix, int iy, int iz) const

virtual long **getImage_long** (int ix) const

virtual long **getImage_long** (int ix, int iy) const

virtual long **getImage_long** (int ix, int iy, int iz) const

virtual unsigned long
getImage_unsigned_long (int ix) const

virtual unsigned long
getImage_unsigned_long (int ix, int iy) const

virtual unsigned long
getImage_unsigned_long (int ix, int iy, int iz) const

virtual float **getImage_float** (int ix) const

virtual float **getImage_float** (int ix, int iy) const

virtual float **getImage_float** (int ix, int iy, int iz) const

virtual double **getImage_double** (int ix) const

virtual double **getImage_double** (int ix, int iy) const

virtual double **getImage_double** (int ix, int iy, int iz) const

virtual long double
getImage_long_double (int ix) const

virtual long double
getImage_long_double (int ix, int iy) const

virtual long double
getImage_long_double (int ix, int iy, int iz) const

virtual void **setImage** (int ix, Number v)

virtual void **setImage** (int ix, int iy, Number v)

virtual void **setImage** (int ix, int iy, int iz, Number v)

virtual void **setImage** (int ix, char v)

virtual void **setImage** (int ix, int iy, char v)

virtual void **setImage** (int ix, int iy, int iz, char v)

virtual void **setImage** (int ix, signed char v)

virtual void **setImage** (int ix, int iy, signed char v)

virtual void **setImage** (int ix, int iy, int iz, signed char v)

virtual void **setImage** (int ix, unsigned char v)

virtual void **setImage** (int ix, int iy, unsigned char v)

virtual void **setImage** (int ix, int iy, int iz, unsigned char v)

virtual void **setImage** (int ix, short v)

virtual void **setImage** (int ix, int iy, short v)

virtual void **setImage** (int ix, int iy, int iz, short v)

virtual void **setImage** (int ix, unsigned short v)

virtual void **setImage** (int ix, int iy, unsigned short v)

virtual void **setImage** (int ix, int iy, int iz, unsigned short v) virtual void **setImage** (int ix, int v)

virtual void **setImage** (int ix, int iy, int v)

virtual void **setImage** (int ix, int iy, int iz, int v)

virtual void **setImage** (int ix, unsigned int v)

virtual void **setImage** (int ix, int iy, unsigned int v)

virtual void **setImage** (int ix, int iy, int iz, unsigned int v)

virtual void **setImage** (int ix, long v)

virtual void **setImage** (int ix, int iy, long v)

virtual void **setImage** (int ix, int iy, int iz, long v)

virtual void **setImage** (int ix, unsigned long v)

virtual void **setImage** (int ix, int iy, unsigned long v)

virtual void **setImage** (int ix, int iy, int iz, unsigned long v)

virtual void **setImage** (int ix, float v)

virtual void **setImage** (int ix, int iy, float v)

virtual void **setImage** (int ix, int iy, int iz, float v)

virtual void **setImage** (int ix, double v)

virtual void **setImage** (int ix, int iy, double v)

virtual void **setImage** (int ix, int iy, int iz, double v)

virtual void **setImage** (int ix, long double v)

virtual void **setImage** (int ix, int iy, long double v)

virtual void **setImage** (int ix, int iy, int iz, long double v)

virtual ImageField*
new_proj_x (int lower, int upper, int mode)

virtual ImageField*
new_proj_y (int lower, int upper, int mode)

virtual ImageField*
new_proj_z (int lower, int upper, int mode)

virtual ImageField*
new_proj_x_avg (int lower, int upper)

virtual ImageField*
new_proj_y_avg (int lower, int upper)

virtual ImageField*
new_proj_z_avg (int lower, int upper)

virtual ImageField*
new_proj_x_max (int lower, int upper)

virtual ImageField*
new_proj_y_max (int lower, int upper)

virtual ImageField*
new_proj_z_max (int lower, int upper)

virtual ImageField*

new_cropped (const int *lattice_box)

virtual ImageField*
new_imbedded (const int *lattice_box, double background)

virtual ImageField*
new_diffused (double diff_coef, int num_iters)

112.2.1

virtual ImageField* **new_by_interpol** (const ImageBase& grid, const MapDef& mapping, int precision_level, const PrecisionChoice& pc, int debug) const

generate field by interpolation

In file ../Field/image_field.hh:8655

generate field by interpolation

Given a value $I(p)$ (e.g. a measured intensity) at a point p, the intensity can be estimated from an interpolation function of several components, that is, $I(p) = c_i * f_i(p)$. Let the form of the interpolation function remain constant and let the coefficients c vary, depending on the region that has center q . Then $I^q(p) = c_i^q * f_i(p)$. For a fixed stencil of points p_i , suppressing the writing of q ,

$$I_i = I(p_i) = c_j * f_j(p_i) \equiv c_j * F_{ji} \, .$$

Since F_{ji} does not depend on q , for a given stencil on a lattice with uniform spacing, there is a unique matrix F for a lattice. The coefficients c^q can be calculated using the inverse of F ,

$$I_j^q (F^{-1})_{ji} = c_i^q \, .$$

If the information available is not the intensity at a point, but rather, the intensity averaged over a box b, then we can write

$$\langle I^q \rangle b_i = c_j^q * \left\langle f_j \right\rangle b_i \equiv c_j^q * G_{ji} \, .$$

The coefficients c^q are the same for both pointwise intensity and box-averaged intensity. The intensity at a point r would then be given by

$$I(r) = c_i^q * f_i(r)$$

and for a box v

$$\langle I \rangle_v = c_i^q * \langle f_i \rangle_v \, .$$

It is assumed that the field values are an average over the area (or volume) of the pixel (or voxel), rather than assuming that the value represents the value at the center of the element.

precision_level == PRECISION_LEVEL1 use value of nearest point

precision_level == PRECISION_LEVEL2 use interpolated value from mapping domain

precision_level == PRECISION_LEVEL3 convert final set of points to interpolated pixels (voxels)

Parameters: `precision_level` precision of interpolation

113 class **NewImageField**

Creates new ImageFieldTyped<T> pointers.

In file ../Field/image_field.hh:8903

Public Members

113.1 **static methods** 214

Creates new ImageFieldTyped<T> pointers.

The implementation uses the templated class ImageFieldTyped<T> but the public interface is at the more basic level of an untyped image field.

Author:	Alan Louis Scheinine
Version:	`$Id: image_field.hh,v 1.8 2002/04/18 23:38:14 alan Exp $`

113.1 static methods

Names

static ImageField*
newImageField (int image_data_type)

static ImageField*
newImageField (int image_data_type, int dimension,
const int *lattice_bounds, const double *aspect_ratio)

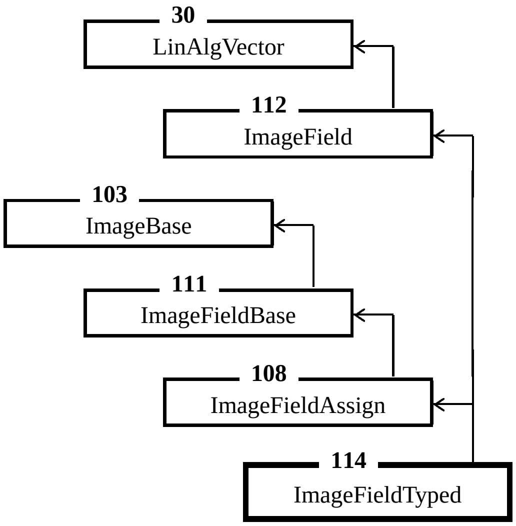
114

template<class T> class **ImageFieldTyped** : public ImageFieldAssign<T>, virtual public ImageField

A basic field with a specified (templated) numerical type.

In file ../Field/image_field_typed.hh:8973

Inheritance



Public Members

114.1	**type definitions**	216
114.4	**usual methods**	216
114.5	**methods**	217
114.6	**static methods**	217
114.7	**Arithmetic methods.**	218
114.8	**virtual functions of LinAlgVector**	220
114.9	**virtual functions of ImageField**	221
114.10	**templated methods**	225

Protected Members

114.3	**copy and assignment helper methods**	225

Private Members

114.2 **helper methods** 226

A basic field with a specified (templated) numerical type.

Implements the virtual functions declared in ImageField and virtual functions declared in LinAlgVector.

Author: Alan Louis Scheinine
Version: `$Id: image_field_typed.hh,v 1.13 2002/04/18 23:38:14 alan Exp $`

114.1

type definitions

Names

typedef Subscript **size_type**

typedef T **value_type**

typedef T **element_type**

typedef T* **pointer**

typedef T* **iterator**

typedef T& **reference**

typedef const T* **const_iterator**

typedef const T& **const_reference**

114.4

usual methods

Names

ImageFieldTyped () *default constructor*

ImageFieldTyped (int dimension, const int *lattice_bounds, const double *aspect_ratio, const T& value = T(0))
constructor

ImageFieldTyped (const ImageFieldTyped<T>& object_in)
copy constructor

ImageFieldTyped (const ImageFieldAssign<T>& object_in)
constructor

template<class U>

ImageFieldTyped (const ImageFieldBase<U>& object_in)
constructor

ImageFieldTyped<T> &
operator= (const ImageFieldTyped<T>& object_in)
assignment

ImageFieldTyped<T> &
operator= (const ImageFieldAssign<T>& object_in)
assignment

template<class U> ImageFieldTyped<T> &
operator= (const ImageFieldBase<U>& object_in)
assignment

ImageFieldTyped<T> &
operator= (const Number& scalar)
assignment

virtual **~ImageFieldTyped** () *destructor*

114.5 methods

Names

inline ImageField*
newImageField () const

inline ImageField*
newImageField (int dimension, const int *lattice_bounds, const double *aspect_ratio) const

inline ImageField*
cloneImageField () const

114.6 static methods

Names

inline static ImageFieldTyped<T> &
cast_to_self_type (ImageField& in)

inline static const ImageFieldTyped<T> &
cast_to_self_type (const ImageField& in)

static int **getStaticDataType** ()

114.7

Arithmetic methods.

Names

inline ImageFieldTyped<T> &
operator+= (const Number& scalar)

inline ImageFieldTyped<T> &
operator-= (const Number& scalar)

inline ImageFieldTyped<T> &
operator*= (const Number& scalar)

inline ImageFieldTyped<T> &
operator/= (const Number& scalar)

inline ImageFieldTyped<T> &
operator+= (char scalar)

inline ImageFieldTyped<T> &
operator-= (char scalar)

inline ImageFieldTyped<T> &
operator*= (char scalar)

inline ImageFieldTyped<T> &
operator/= (char scalar)

inline ImageFieldTyped<T> &
operator+= (signed char scalar)

inline ImageFieldTyped<T> &
operator-= (signed char scalar)

inline ImageFieldTyped<T> &
operator*= (signed char scalar)

inline ImageFieldTyped<T> &
operator/= (signed char scalar)

inline ImageFieldTyped<T> &
operator+= (unsigned char scalar)

inline ImageFieldTyped<T> &
operator-= (unsigned char scalar)

inline ImageFieldTyped<T> &
operator*= (unsigned char scalar)

inline ImageFieldTyped<T> &
operator/= (unsigned char scalar)

inline ImageFieldTyped<T> &
operator+= (short scalar)

inline ImageFieldTyped<T> &
operator-= (short scalar)

inline ImageFieldTyped<T> &
operator*= (short scalar)

inline ImageFieldTyped<T> &

operator/= (short scalar)

inline ImageFieldTyped<T> &
operator+= (unsigned short scalar)

inline ImageFieldTyped<T> &
operator-= (unsigned short scalar)

inline ImageFieldTyped<T> &
operator*= (unsigned short scalar)

inline ImageFieldTyped<T> &
operator/= (unsigned short scalar)

inline ImageFieldTyped<T> &
operator+= (int scalar)

inline ImageFieldTyped<T> &
operator-= (int scalar)

inline ImageFieldTyped<T> &
operator*= (int scalar)

inline ImageFieldTyped<T> &
operator/= (int scalar)

inline ImageFieldTyped<T> &
operator+= (unsigned int scalar)

inline ImageFieldTyped<T> &
operator-= (unsigned int scalar)

inline ImageFieldTyped<T> &
operator*= (unsigned int scalar)

inline ImageFieldTyped<T> &
operator/= (unsigned int scalar)

inline ImageFieldTyped<T> &
operator+= (long scalar)

inline ImageFieldTyped<T> &
operator-= (long scalar)

inline ImageFieldTyped<T> &
operator*= (long scalar)

inline ImageFieldTyped<T> &
operator/= (long scalar)

inline ImageFieldTyped<T> &
operator+= (unsigned long scalar)

inline ImageFieldTyped<T> &
operator-= (unsigned long scalar)

inline ImageFieldTyped<T> &
operator*= (unsigned long scalar)

inline ImageFieldTyped<T> &
operator/= (unsigned long scalar)

inline ImageFieldTyped<T> &
operator+= (float scalar)

inline ImageFieldTyped<T> &

operator-= (float scalar)

inline ImageFieldTyped<T> &
operator*= (float scalar)

inline ImageFieldTyped<T> &
operator/= (float scalar)

inline ImageFieldTyped<T> &
operator+= (double scalar)

inline ImageFieldTyped<T> &
operator-= (double scalar)

inline ImageFieldTyped<T> &
operator*= (double scalar)

inline ImageFieldTyped<T> &
operator/= (double scalar)

inline ImageFieldTyped<T> &
operator+= (long double scalar)

inline ImageFieldTyped<T> &
operator-= (long double scalar)

inline ImageFieldTyped<T> &
operator*= (long double scalar)

inline ImageFieldTyped<T> &
operator/= (long double scalar)

template<class U>inline ImageFieldTyped<T> &
operator+= (const ImageFieldTyped<U>& A)

template<class U>inline ImageFieldTyped<T> &
operator-= (const ImageFieldTyped<U>& A)

template<class U>inline ImageFieldTyped<T> &
operator*= (const ImageFieldTyped<U>& A)

template<class U>inline ImageFieldTyped<T> &
operator/= (const ImageFieldTyped<U>& A)

Partial specializations of templated functions do not work in gcc so need to list many functions explicitly to avoid ambiguities; that is, gcc does not favor the more specific partial specialization and does not favor the case with less conversion of the argument.

114.8

virtual functions of LinAlgVector

Names

inline LinAlgVector&
operator= (const LinAlgScalar& las)

inline LinAlgVector&

operator- ()

inline LinAlgVector*
newLinAlgVector () const

inline LinAlgVector*
clone () const

inline LinAlgVector&
operator= (const LinAlgVector& lav)

inline LinAlgVector&
operator+= (const LinAlgScalar& las)

inline LinAlgVector&
operator-= (const LinAlgScalar& las)

inline LinAlgVector&
operator*= (const LinAlgScalar& las)

inline LinAlgVector&
operator/= (const LinAlgScalar& las)

inline LinAlgVector&
operator+= (const LinAlgVector& lav)

inline LinAlgVector&
operator-= (const LinAlgVector& lav)

inline LinAlgVector&
operator*= (const LinAlgVector& lav)

inline std::ostream&
put (std::ostream& s) const

inline std::istream&
get (std::istream& s)

114.9

virtual functions of ImageField

Names

ImageField& **operator=** (const ImageField& object_in)

inline void **setBounds** (const int *lattice_bounds)

inline int **getBounds** (int i) const

inline void **setAspect** (const double *aspect_ratio)

inline double **getAspect** (int i) const

inline void **setDimension** (unsigned char dimension)

inline unsigned char
getDimension () const

inline const int* const
getBounds_array () const

inline const double* const

getAspect_array () const

inline bool **find_indices_nearest** (const double* const coord, int* const lattice_site,
double* const location) const

inline bool **find_indices_nearest** (const double* const coord, int* const lattice_site,
double* const location,
const signed char* field_pos) const

inline const ImageBase*
getImageBase () const

ImageField* **new_extend_by_two** () const

int **make_stencil_rhs** (StencilVector_ref rhs, const StencilSites& sites,
const int* lattice_site, bool check_bounds) const

int **check_template_with_field** (int dimension_must_be,
const int* lattice_site, int extent) const

ImageField* **new_by_interpol** (const ImageBase& grid, const MapDef& mapping,
int precision_level, const PrecisionChoice& pc,
int debug) const

inline int **getDataType** () const

inline int **toDataType** (const char* type_in)

inline Number**getImage_Number** (int ix) const

inline Number**getImage_Number** (int ix, int iy) const

inline Number**getImage_Number** (int ix, int iy, int iz) const

inline char **getImage_char** (int ix) const

inline char **getImage_char** (int ix, int iy) const

inline char **getImage_char** (int ix, int iy, int iz) const

inline signed char
getImage_signed_char (int ix) const

inline signed char
getImage_signed_char (int ix, int iy) const

inline signed char
getImage_signed_char (int ix, int iy, int iz) const

inline unsigned char
getImage_unsigned_char (int ix) const

inline unsigned char
getImage_unsigned_char (int ix, int iy) const

inline unsigned char
getImage_unsigned_char (int ix, int iy, int iz) const

inline short **getImage_short** (int ix) const

inline short **getImage_short** (int ix, int iy) const

inline short **getImage_short** (int ix, int iy, int iz) const

inline unsigned short

getImage_unsigned_short (int ix) const

inline unsigned short
getImage_unsigned_short (int ix, int iy) const

inline unsigned short
getImage_unsigned_short (int ix, int iy, int iz) const

inline int **getImage_int** (int ix) const

inline int **getImage_int** (int ix, int iy) const

inline int **getImage_int** (int ix, int iy, int iz) const

inline unsigned int
getImage_unsigned_int (int ix) const

inline unsigned int
getImage_unsigned_int (int ix, int iy) const

inline unsigned int
getImage_unsigned_int (int ix, int iy, int iz) const

inline long **getImage_long** (int ix) const

inline long **getImage_long** (int ix, int iy) const

inline long **getImage_long** (int ix, int iy, int iz) const

inline unsigned long
getImage_unsigned_long (int ix) const

inline unsigned long
getImage_unsigned_long (int ix, int iy) const

inline unsigned long
getImage_unsigned_long (int ix, int iy, int iz) const

inline float **getImage_float** (int ix) const

inline float **getImage_float** (int ix, int iy) const

inline float **getImage_float** (int ix, int iy, int iz) const

inline double **getImage_double** (int ix) const

inline double **getImage_double** (int ix, int iy) const

inline double **getImage_double** (int ix, int iy, int iz) const

inline long double
getImage_long_double (int ix) const

inline long double
getImage_long_double (int ix, int iy) const

inline long double
getImage_long_double (int ix, int iy, int iz) const

inline void **setImage** (int ix, Number v)

inline void **setImage** (int ix, int iy, Number v)

inline void **setImage** (int ix, int iy, int iz, Number v)

inline void **setImage** (int ix, char v)

inline void **setImage** (int ix, int iy, char v)

inline void **setImage** (int ix, int iy, int iz, char v) inline void **setImage** (int ix, signed char v)

inline void **setImage** (int ix, int iy, signed char v)

inline void **setImage** (int ix, int iy, int iz, signed char v)

inline void **setImage** (int ix, unsigned char v)

inline void **setImage** (int ix, int iy, unsigned char v)

inline void **setImage** (int ix, int iy, int iz, unsigned char v)

inline void **setImage** (int ix, short v)

inline void **setImage** (int ix, int iy, short v)

inline void **setImage** (int ix, int iy, int iz, short v)

inline void **setImage** (int ix, unsigned short v)

inline void **setImage** (int ix, int iy, unsigned short v)

inline void **setImage** (int ix, int iy, int iz, unsigned short v)

inline void **setImage** (int ix, int v)

inline void **setImage** (int ix, int iy, int v)

inline void **setImage** (int ix, int iy, int iz, int v)

inline void **setImage** (int ix, unsigned int v)

inline void **setImage** (int ix, int iy, unsigned int v)

inline void **setImage** (int ix, int iy, int iz, unsigned int v)

inline void **setImage** (int ix, long v)

inline void **setImage** (int ix, int iy, long v)

inline void **setImage** (int ix, int iy, int iz, long v)

inline void **setImage** (int ix, unsigned long v)

inline void **setImage** (int ix, int iy, unsigned long v)

inline void **setImage** (int ix, int iy, int iz, unsigned long v)

inline void **setImage** (int ix, float v)

inline void **setImage** (int ix, int iy, float v)

inline void **setImage** (int ix, int iy, int iz, float v)

inline void **setImage** (int ix, double v)

inline void **setImage** (int ix, int iy, double v)

inline void **setImage** (int ix, int iy, int iz, double v)

inline void **setImage** (int ix, long double v)

inline void **setImage** (int ix, int iy, long double v)

inline void **setImage** (int ix, int iy, int iz, long double v)

inline ImageField*
new_proj_x (int lower, int upper, int mode)

inline ImageField*
new_proj_y (int lower, int upper, int mode)

inline ImageField*

new_proj_z (int lower, int upper, int mode)

inline ImageField*
new_proj_x_avg (int lower, int upper)

inline ImageField*
new_proj_y_avg (int lower, int upper)

inline ImageField*
new_proj_z_avg (int lower, int upper)

inline ImageField*
new_proj_x_max (int lower, int upper)

inline ImageField*
new_proj_y_max (int lower, int upper)

inline ImageField*
new_proj_z_max (int lower, int upper)

inline ImageField*
new_cropped (const int *lattice_box)

inline ImageField*
new_imbedded (const int *lattice_box, double background)

inline ImageField*
new_diffused (double diff_coef, int num_iters)

114.10

templated methods

Names

template<class U> U
getImage_primitive (int ix) const

template<class U> U
getImage_primitive (int ix, int iy) const

template<class U> U
getImage_primitive (int ix, int iy, int iz) const

template<class U> void
setImage_primitive (int ix, U v)

template<class U> void
setImage_primitive (int ix, int iy, U v)

template<class U> void
setImage_primitive (int ix, int iy, int iz, U v)

114.3

copy and assignment helper methods

Names

inline void **convert** (const ImageFieldTyped<T>& object_in)
Copy of data members.

inline void **convert_tree** (const ImageFieldTyped<T>& object_in)
Call convert_tree on each parent class then call convert.

114.2 helper methods

Names

inline int **bnds** (int i) const

inline double **spct** (int i) const

inline int **dmsn** () const

inline const int* **bnds_array** () const

inline const double* **spct_array** () const

115

template<class T>inline ImageFieldTyped<T> **operator+** (const Number& A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10120

116

template<class T>inline ImageFieldTyped<T> **operator-** (const Number& A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10125

117

template<class T>inline ImageFieldTyped<T> **operator*** (const Number& A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10130

118

template<class T>inline ImageFieldTyped<T> **operator+** (char A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10135

119

template<class T>inline ImageFieldTyped<T> **operator-** (char A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10140

120

template<class T>inline ImageFieldTyped<T> **operator*** (char A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10145

121

template<class T>inline ImageFieldTyped<T> **operator+** (signed char A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10150

122

template<class T>inline ImageFieldTyped<T> **operator-** (signed char A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10155

123

template<class T>inline ImageFieldTyped<T> **operator*** (signed char A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10160

124

template<class T>inline ImageFieldTyped<T> **operator+** (unsigned char A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10165

125

template<class T>inline ImageFieldTyped<T> **operator-** (unsigned char A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10170

126

template<class T>inline ImageFieldTyped<T> **operator*** (unsigned char A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10175

127

template<class T>inline ImageFieldTyped<T> **operator+** (short A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10180

128

template<class T>inline ImageFieldTyped<T> **operator-** (short A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10185

129

template<class T>inline ImageFieldTyped<T> **operator*** (short A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10190

130

template**<**class T**>**inline ImageFieldTyped**<**T**>** **operator+** (unsigned short A, const ImageFieldTyped**<**T**>**& B)

In file ../Field/image_field_typed.hh:10195

131

template<class T>inline ImageFieldTyped<T> **operator-** (unsigned short A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10200

132

template<class T>inline ImageFieldTyped<T> **operator*** (unsigned short A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10205

133

template<class T>inline ImageFieldTyped<T> **operator+** (int A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10210

134

template<class T>inline ImageFieldTyped<T> **operator-** (int A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10215

135

template<class T>inline ImageFieldTyped<T> **operator*** (int A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10220

136

template<class T>inline ImageFieldTyped<T> **operator+** (unsigned int A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10225

137

template<class T>inline ImageFieldTyped<T> **operator-** (unsigned int A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10230

138

template<class T>inline ImageFieldTyped<T> **operator*** (unsigned int A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10235

139

template<class T>inline ImageFieldTyped<T> **operator+** (long A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10240

140

template<class T>inline ImageFieldTyped<T> **operator-** (long A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10245

141

template<class T>inline ImageFieldTyped<T> **operator*** (long A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10250

142

template<class T>inline ImageFieldTyped<T> **operator+** (unsigned long A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10255

143

template<class T>inline ImageFieldTyped<T> **operator-** (unsigned long A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10260

144

template<class T>inline ImageFieldTyped<T> **operator*** (unsigned long A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10265

145

template<class T>inline ImageFieldTyped<T> **operator+** (float A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10270

146

template<class T>inline ImageFieldTyped<T> **operator-** (float A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10275

147

template<class T>inline ImageFieldTyped<T> **operator*** (float A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10280

148

template<class T>inline ImageFieldTyped<T> **operator+** (double A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10285

149

template<class T>inline ImageFieldTyped<T> **operator-** (double A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10290

150

template<class T>inline ImageFieldTyped<T> **operator*** (double A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10295

151

template<class T>inline ImageFieldTyped<T> **operator+** (long double A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10300

152

template<class T>inline ImageFieldTyped<T> **operator-** (long double A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10305

153

template<class T>inline ImageFieldTyped<T> **operator*** (long double A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10310

154

template<class T>inline ImageFieldTyped<T> **operator+** (const LinAlgScalar& las, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10315

155

template<class T>inline ImageFieldTyped<T> **operator-** (const LinAlgScalar& las, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10320

156

template<class T>inline ImageFieldTyped<T> **operator*** (const LinAlgScalar& las, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10325

157

template<class T>inline ImageFieldTyped<T> **operator+** (const ImageFieldTyped<T>& A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10330

158

template<class T>inline ImageFieldTyped<T> **operator-** (const ImageFieldTyped<T>& A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10337

159

template<class T>inline ImageFieldTyped<T> **operator*** (const ImageFieldTyped<T>& A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10344

160

template<class T> std::ostream& **operator<<** (std::ostream &s, const ImageFieldTyped<T> &A)

ImageFieldTyped write to standard output.

In file ../Field/image_field_typed.hh:10351

ImageFieldTyped write to standard output.

161

template<class T> std::istream& **operator>>** (std::istream &s, ImageFieldTyped<T> &A)

ImageFieldTyped read from standard input.

In file ../Field/image_field_typed.hh:10355

ImageFieldTyped read from standard input.

162

class **ImageFieldAlgorithms**

Simple algorithms applied to a simple field

In file ../Field/image_field_algorithms.hh:10368

Public Members

162.1 **static methods** 274

Simple algorithms applied to a simple field

162.1

static methods

Names

static ImageField*
new_proj_x (int image_data_type, const ImageField& img_fld,
int lower, int upper, int mode)

static ImageField*
new_proj_y (int image_data_type, const ImageField& img_fld,
int lower, int upper, int mode)

static ImageField*
new_proj_z (int image_data_type, const ImageField& img_fld,
int lower, int upper, int mode)

static ImageField*
new_proj_x_avg (int image_data_type, const ImageField& img_fld,
int lower, int upper)

static ImageField*
new_proj_y_avg (int image_data_type, const ImageField& img_fld,
int lower, int upper)

static ImageField*
new_proj_z_avg (int image_data_type, const ImageField& img_fld,
int lower, int upper)

static ImageField*
new_proj_x_max (int image_data_type, const ImageField& img_fld,
int lower, int upper)

static ImageField*

new_proj_y_max (int image_data_type, const ImageField& img_fld,
int lower, int upper)

static ImageField*
new_proj_z_max (int image_data_type, const ImageField& img_fld,
int lower, int upper)

static ImageField*
new_cropped (int image_data_type, const ImageField& img_fld,
const int *lattice_box)

static ImageField*
new_imbedded (int image_data_type, const ImageField& img_fld,
const int *lattice_box, double background)

static ImageField*
new_diffused (int image_data_type, const ImageField& img_fld,
double diff_coef, int num_iters)

163

class **FieldInterpolHelper**

helper methods for interpolation algorithms

In file ../Field/field_interpol_algorithms.hh:10481

Public Members

163.4 **usual methods** 276

Protected Members

163.1 **copy and assignment helper methods** 276

163.2 **data** 277

163.3 **initialization** 277

helper methods for interpolation algorithms

Author: Alan Louis Scheinine
Version: `$Id: field_interpol_algorithms.hh,v 1.10 2002/04/05 22:25:39 alan Exp $`

163.4

usual methods

Names

FieldInterpolHelper () *default constructor*

163.1

copy and assignment helper methods

Names

void **convert** (const FieldInterpolHelper& object_in)
Copy of data members.

void **convert_tree** (const FieldInterpolHelper& object_in)
Call convert_tree on each parent class then call convert.

163.2

data

Names

StencilMatrix*	**nvrs**
StencilVector	**rhs**
StencilVector	**coefs**
StencilTerms*	**stencil**

163.3

initialization

Names

void	**set_defaults** ()

164 class **FieldInterpolAlgorithms**

interpolation algorithms for a regular grid

In file ../Field/field_interpol_algorithms.hh:10568

Public Members

164.1 **static methods** 278

Private Members

164.2 **private static methods** 279

interpolation algorithms for a regular grid

Author: Alan Louis Scheinine
Version: `$Id: field_interpol_algorithms.hh,v 1.10 2002/04/05 22:25:39 alan Exp $`

164.1 static methods

Names

static int **check_template_with_field** (const ImageField& fldntrpl,
int dimension_must_be,
const int* lattice_site, int extent)

static ImageField*
new_extend_by_two (int image_data_type, const ImageField& fldntrpl)

static int **make_stencil_rhs** (const ImageField& fldntrpl, StencilVector_ref rhs,
const StencilSites& sites, const int* lattice_site,
bool check_bounds)

164.1.1 static ImageField*
new_by_interpol (int image_data_type, const ImageField& fldntrpl,
const ImageBase& grid, const MapDef& mapping,
int precision_level, const PrecisionChoice& pc,
int debug)
Constructor using a mapping of a grid. 279

static int **point_to_voxel** (int image_data_type, const ImageField* field_pnt,
ImageField* field_out, const PrecisionChoice& pc)

164.1.1

static ImageField* **new_by_interpol** (int image_data_type, const ImageField& fldntrpl, const ImageBase& grid, const MapDef& mapping, int precision_level, const PrecisionChoice& pc, int debug)

Constructor using a mapping of a grid.

In file ../Field/field_interpol_algorithms.hh:10597

Constructor using a mapping of a grid.

The grid points defined by

```
ImageBase(int dimension, int lattice_bounds[3], double aspect_ratio[3])
```

are mapped onto fldntrpl.

164.2

private static methods

Names

static	int	**extend_helper** (int image_data_type, const ImageField& fldntrpl, FieldInterpolHelper* helper, int ix, int iy, int iz, int cx, int cy, int cz, float *pt_intensity)
static	int	**make_stencil_rhs_too** (int dimen, const ImageField& fld, StencilVector_ref rhs, const StencilSites& sites, const int* lattice_site, bool check_bounds)
static	void	**new_by_interpol_free** (ImageField* field_extended, vector<ImageField*>& all_field_pnt)
static	void	**new_by_interpol_free** (ImageField* field_extended, vector<ImageField*>& all_field_pnt, ImageField* field_out)

165

class **RegridBrick**

Changes the resolution of an image.

In file ../Utils/regrid_brick.hh:10684

Public Members

165.1 **usual methods** 281

165.2 **static methods** 281

Protected Members

165.3 static ImageField*
new_brick (int image_data_type, const ImageField& fldin,
const ImageBase& grid, const PrecisionChoice& pc,
int debug)
Cannot change both aspect_ratio and lattice_bounds so this method is not public 281

Changes the resolution of an image.

```
The PrecisionChoice can be any of five pairs:

stencil  basis

1        1

1        2

1        9

2        1

2        2

2        9

3        2

3        9
```

Basis 1 is Taylor expansion. Basis 2 are Gaussians. Basis 9 gives bad results and is included only for testing.

Author: Alan Louis Scheinine
Version: `$Id: regrid_brick.hh,v 1.9 2002/04/23 23:15:19 alan Exp $`

165.1 usual methods

Names

	RegridBrick ()	*default constructor*
	RegridBrick (const RegridBrick& object_in)	*copy constructor*
RegridBrick&	**operator=** (const RegridBrick& object_in)	*assignment*
	˜RegridBrick ()	*destructor*

165.2 static methods

Names

static ImageField*
new_brick_from_aspect (int image_data_type,
const ImageField& fldin,
const double* aspect_ratio,
const PrecisionChoice& pc, int debug)

static ImageField*
new_brick_from_bounds (int image_data_type,
const ImageField& fldin,
const int* lattice_bounds,
const PrecisionChoice& pc, int debug)

165.3 static ImageField* **new_brick** (int image_data_type, const ImageField& fldin, const ImageBase& grid, const PrecisionChoice& pc, int debug)

Cannot change both aspect_ratio and lattice_bounds so this method is not public

In file ../Utils/regrid_brick.hh:10725

Cannot change both aspect_ratio and lattice_bounds so this method is not public

166

class **GridSlice**

Makes a two-dimensional grid for a slice of a three-dimensional grid.

In file ../Utils/grid_slice.hh:10773

Public Members

166.1 **usual methods** 282

166.2 **methods** 283

166.3 **static methods** 283

Private Members

166.4 **data** 283

Makes a two-dimensional grid for a slice of a three-dimensional grid.

A copy or assignment of this class is a shallow copy of a counted pointer of the internal data that is the extended field derived from the original field.

Author: Alan Louis Scheinine
Version: `$Id: grid_slice.hh,v 1.3 2002/04/07 15:49:07 alan Exp $`

166.1

usual methods

Names

	GridSlice ()	*default constructor*
	GridSlice (const ImageField& field_in)	*constructor*
	GridSlice (const GridSlice& object_in)	*copy constructor*
GridSlice&	**operator=** (const GridSlice& object_in)	*assignment*
virtual	**˜GridSlice** ()	*destructor*

166.2 methods

Names

void **setSource** (const ImageField& field_in)

ImageField* **newImageField** (const double* position, const double* xdir,
const double* ydir, const ImageBase& grid,
double thickness, int precision_level,
const PrecisionChoice& pc, int debug) const

166.3 static methods

Names

inline static void
cross (const double* a, const double* b, double* c)

166.4 data

Names

int **_image_data_type**

ObjVar<ImageField>
_field_extended

167

Documentation

Names

167.1 **Basic numbers and numerical vectors.** 284

167.2 **Stencils.** 286

167.1

Basic numbers and numerical vectors.

LimitRange The class **template<class T> class** LimitRange (→ 4, *page* 14) is used for the conversion between primitive numerical types. The role is similar to a static cast. Since it operates on one number at a time, the conversion is not efficient. Nonetheless, the class may be useful for type conversion of fields between different steps of processing while avoiding compiler warnings. Here are some examples of the functions

```
static T limit_range(char s);
static T limit_range(unsigned int s);
static T limit_range(double s);
```

The primitive numerical types that can be used for the function parameter or return value are *char, signed char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, float, double* and *long double*.

TNTVect The class **template<class T> class** TNTVect (→ 40, *page* 66) inherits from the TNT class **Vector<T>**. A few more functions have been added and some functions have been changed to increase efficiency. This is a vector for primitive numerical types rather than a vector of an arbitrary class. Note that the size does not change when allocation is done beyond the range of the internal array. Such an assignment is an error. The size is set when the class is constructed or by using **newsize(int)**.

cast_to_self_type The global function **template<class T, class U> T& cast_to_self_type(U& in)** (in namespace CastToSelfType (→ 2, *page* 12)) is not specifically related to numbers, but it is mentioned here because it is used in the class Number (→ 5, *page* 15). In general, this class is used whenever a base class is used to generalize an algorithm using virtual functions that is actually implemented using derived classes.

Though not specifically related to numbers, this function has been motivated by the use of overloaded arithmetic operators, as described for the class Number (→ 5, *page* 15).

Number The class Number (→ 5, *page* 15) represents one number without specifying the type. It can be useful in numerical algorithms in which the program developer does not want use templating to generalize the algorithm. The actual number is not stored in this class but in the derived class NumberTyped (→ 11, *page* 24). In a typical application, a pointer to a base class is actually a pointer to a derived class.

This class defines the virtual functions

```
  virtual Number* newNumber() const
  virtual Number* clone() const
```

Though the return type is declared to be **Number***, the pointers actually point to a derived type that is the same as the object on which the functions are called. **newNumber()** creates a default instantation whereas **clone()** assigns the same value as the object on which the function is called. The assignment operator

```
virtual Number& operator=(const Number& object_in)
```

is also virtual so derived types can copy all members.

This class has a protected data member **Number* rep** which is non-zero if the most derived class of the instantation is actually the base class, in which case, **rep** should point to a derived type. Due to the overhead, this class is not practical for fields of numbers. Nevertheless, it can be very useful when groups of algorithms, implemented in terms of virtual functions that do not need to specify the numerical type of the underlying field, occasionally need to refer to a number without specifying the type.

The motivation for the particular structure of this class is the following. Some arithmetic operations need to return a value by copying, such as operations that generate a temporary variable. Unlike returning by pointer or reference, copying a base class means losing the information of the derived class. The solution is to have a pointer in the base class that is non-zero if the most derived class of a pointer (or reference) is actually the base class. The pointer held in the base class has the type of the base class but is actually a pointer to a derived class.

In other words, a base class does not define the implementations of various virtual functions, a derived class is needed. In the case that the base class must be copied, the base class has a pointer to a derived class (declared as type **Number*** but always pointing to a derived class). For all virtual functions, the base class calls the same function on its pointer to a derived class.

This class also servers as an interface to LimitRange (→ 4, *page* 14). The virtual functions *type* getNumber_*typename*() are defined for almost every possible numerical type. To be more specific, below are some examples of the definitions.

```
virtual signed char getNumber_signed_char() const
virtual unsigned short getNumber_unsigned_short() const
virtual signed int getNumber_int() const
virtual double getNumber_double() const
```

Independent of the underlying type of **Number**, the user can have the value as any arbitrary type.

It is also possible to set the number from any type. Due to function overloading of parameters, the function name is simply **setNumber**. A few examples will be given to clarify the idea.

```
virtual void setNumber(signed char v)
virtual void setNumber(unsigned short v)
virtual void setNumber(signed int v)
virtual void setNumber(double v)
```

There are also templated global functions (in namespace GetNumber (→ 10, *page* 23))

```
template <class T> T getNumber(const Number& n);
```

for use inside templated classes. For example, a templated function can use the function **GetNumber::getNumber<T>(n)** and does not need to be as specific as, for example, **n.getNumber_unsigned_short()**.

NumberTyped The class **template <class T> class** NumberTyped (→ 11, *page* 24) inherits from Number (→ 5, *page* 15) and implements the virtual functions of Number (→ 5, *page* 15). This class contains one protected data member: **T _value**. It implements all of the virtual functions of Number (→ 5, *page* 15), using LimitRange (→ 4, *page* 14) when numeric conversions are necessary.

167.2 Stencils.

StencilParams The class StencilParams (→ 88, *page* 147) holds the most basic information about a stencil, whether the stencil has two or three spatial dimensions and number of sites, but does not contain an actual stencil as a data member. The constructor is **StencilParams(int tag_in)** where the tag is either 3, 5, 9, 13, 21, 25, 27, 33 or 57. These tag values are also the number of sites for the stencils, the first two are one-dimensional, the second three are two-dimesional and the last three are three-dimensional. The functions **int getTag()**, **int getSize()** and **int getDimension()** return the basic information.

StencilSitesTag The class StencilSitesTag (→ 87, *page* 145) contains one integer that specifies the stencil configuration. A class is used rather than just using an integer in order to avoid confusion with the tag that specifies the polynomial terms.

StencilSites The class StencilSites (→ 85, *page* 130) inherits from StencilParams and contains lists of stencil sites. The center point of the stencils are given indices (0, 0, 0) in three dimensions. The one and two dimensional stencils are also centered at zero. The public members **stencil_sites_x**, **stencil_sites_y**, and **stencil_sites_z** are array classes indexed by the stencil site. The size of the arrays correspond to the tag used in the constructor **StencilSites(int tag_in)**. Though public, these arrays are of type **ReadOnlyNumArray<signed char>** and cannot be changed once an object is instantiated.

ArbitrarySites The class ArbitrarySites (→ 86, *page* 137) contains lists of sites. The data members include the number of dimensions (1, 2 or 3) and the number of sites. The public arrays of type **ReadOnlyNumArray<double>** contain the dimensions of the sites. Note that the arrays of class StencilSites have values that are grid indices whereas the arrays of this class are real-valued positions. The functions **int getSize()** and **int getDimension()** return the corresponding information. The constructors

```
ArbitrarySites(int size_in,
               const double* stencil_sites_xx,
               const double* stencil_sites_yy)
ArbitrarySites(int size_in,
               const double* stencil_sites_xx,
               const double* stencil_sites_yy,
               const double* stencil_sites_zz)
ArbitrarySites(const ReadOnlyNumArray<double>& stencil_sites_xx,
               const ReadOnlyNumArray<double>& stencil_sites_yy)
ArbitrarySites(const ReadOnlyNumArray<double>& stencil_sites_xx,
               const ReadOnlyNumArray<double>& stencil_sites_yy,
               const ReadOnlyNumArray<double>& stencil_sites_zz)
```

have argument parameters that are either arrays of double or the class **ReadOnlyNumArray<double>**.

StencilVector The typedef StencilVector (→ 94, *page* 173) defines the class **TNTVect<double>** It has a general-purpose role as a vector that can hold coefficients or the result of vector-matrix multiplication.

StencilTerms Let us consider the value of a scalar field to be a polynomial function of position. The polynomial has fixed coefficents of terms such as x , x^2 , xy^2 , etc. For such a Taylor expansion, the class StencilTerms (→ 91, *page* 156) generates a vector containing the polynomial terms for a given point (x, y, z) using the function **void make_point_terms(const double *location)** and placing the result in the public array **TNTVect<double> terms**. The function **void make_ntgrl_terms(const double *box)** averages the terms over a rectangle or right parallelpiped, placing the result in the same array. This latter function converts the underlying function into an output that might be similar to the output from an actual scan in which the intensity is an average over a certain area or volume. The function **void make_gradient_terms(const double *location)** computes the terms of the gradient and puts the results in the arrays TNTVect<double> xterms, yterms, zterms.

In addition, the class StencilTerms (→ 91, *page* 156) can be constructed so as to use Gaussian weights (approximated by bsplines) rather than Taylor expansion terms.

The Taylor expansion has been define for the following number of terms: 3, 5, 9, 13, 27 or 33. If instead, Gaussian weights are used for the basis functions, the number of terms (which equals the number of stencil sites) can be any of the following: 3, 5, 9, 13, 21, 25, 27, 33 or 57.

StencilMatrix The class StencilMatrix (→ 92, *page* 164) constructs a matrix of stencil terms (polynomial terms) based on the tag given to the constructor **StencilMatrix(int tag_in)**. The value of the field at each point is not used in this class, though the class does use the ‘aspect’ variable of ImageBase (→ 103, *page* 182) to convert x,y,z indices to relative positions in space. The polynomial terms of the matrix are generated by

```
int fill_matrix(const double *position,
                const ImageBase *field,
                int value_mode)
```

The matrix contains polynomial terms using absolute positions rather than relative positions, the conversion is done by passing as a parameter a three component array that gives the absolute position of the center of the stencil. The **field** is used only for the lattice spacing. The parameter **value_mode** can have value POINT_VALUE_MODE or BOX_VALUE_MODE. The latter value indicates that the stencil terms correspond to field values averaged over a box. The function

```
int take_inverse()
```

takes the inverse of the matrix. The function

```
int generateLatticeInverse(const ImageBase* lattice,
                           int value_mode)
```

both fills the matrix and takes the inverse, with the center of the stencil assigned position (0,0,0).

Multiplication between this matrix and a vector of coefficients is done with the following methods.

```
void mat_vec_mult(StencilVector_const_ref stencil_coefs_in,
                  StencilVector_ref stencil_coefs_out) const;
void vec_mat_mult(StencilVector_const_ref stencil_coefs_in,
                  StencilVector_ref stencil_coefs_out) const;
```

ArbitraryMatrix The class ArbitraryMatrix (→ 93, *page* 168) constructs a matrix of stencil terms (polynomial terms) based on arbitrary positions. The field values are not used in this class, though it is assumed that the for each point there is a corresponding field value.

The positions in space and the form of the polynomial are specified in the constructor. The form of the polynomial uses the same tag as the stencil tag, though stencil positions are not relevant to this class. The reason why the stencil tag has meaning for this class is that each tag also corresponds to a specific form of polynomial. The input of the positions can be either a simple array of doubles or can be obtained from various classes that can contain an array, as shown below.

```
ArbitraryMatrix(ArbitrarySites& object_in,
                int tag_in)
ArbitraryMatrix(int num_sites_in,
                const double* stencil_sites_xx,
                const double* stencil_sites_yy,
                int tag_in)
ArbitraryMatrix(int num_sites_in,
                const double* stencil_sites_xx,
                const double* stencil_sites_yy,
                const double* stencil_sites_zz,
                int tag_in)
ArbitraryMatrix(const ReadOnlyNumArray<double>& stencil_sites_xx,
                const ReadOnlyNumArray<double>& stencil_sites_yy,
                int tag_in)
ArbitraryMatrix(const ReadOnlyNumArray<double>& stencil_sites_xx,
                const ReadOnlyNumArray<double>& stencil_sites_yy,
                const ReadOnlyNumArray<double>& stencil_sites_zz,
                int tag_in)
ArbitraryMatrix(const TNTVect<double>& stencil_sites_xx,
                const TNTVect<double>& stencil_sites_yy,
                int tag_in)
ArbitraryMatrix(const TNTVect<double>& stencil_sites_xx,
                const TNTVect<double>& stencil_sites_yy,
                const TNTVect<double>& stencil_sites_zz,
                int tag_in)
```

The polynomial terms of the matrix are generated by

```
int fill_matrix()
```

It is assumed that there are more points than polynomial terms. For this overdetermined system, the calculation of polynomial coefficients uses the square matrix of size equal to the number of terms. This matrix is calculated by either of the following functions.

```
StencilMatrix makeInnerSquared() const
void makeInnerSquared(StencilMatrix& tmp) const
```

The former returns a copy of a StencilMatrix whereas the latter uses a StencilMatrix already allocated. The second method should be more efficient.

The classes StencilMatrix and ArbitraryMatrix are similar since they both contain a matrix indexed by sites and by polynomial terms. However, the two classes are constructed differently. It can be useful to list these differences because they have an impact on various algorithms beyond those in the classes not only StencilMatrix and ArbitraryMatrix. StencilMatrix inherits from StencilSites whereas ArbitraryMatrix inherits from ArbitrarySites. With regard to the sites, the former refers to fixed positions whereas the latter uses arbitrary positions in space.

StencilSites has a tag

```
  int getTag() const
  int getSize() const
  int getDimension() const
```

when constructed with a tag generates stencil_sites_x (y and z) of indices ArbitrarySites

```
  int getSize() const
  int getDimension() const
```

makes a copy of real-valued positions x, y, z

StencilSites does not have a specific xyz positions, xyz positions are generated when StencilMatrix is generated from position (an offset) and field->getAspectArray(). Moreover the site indices are generated by simply giving a tag value.

For ArbitrarySites, the xyz positions must be given. On the other hand, when generating the matrix, the offset and aspect ratio does not need to be specified. A variable to store a tag (which relates to the terms, only) is in ArbitraryMatrix because ArbitrarySites does not need a tag. Since the num rows and num cols is different, there are two new methods in Arbitrary Matrix

```
numTerms()
numSites() == ArbitrarySites::getSize()
```

ImageBase The class ImageBase (→ 103, *page* 182) contains the most basic functions related to an image or field on a regular grid. It does not contain actual data for a field. The constructor is

```
ImageBase(int dimension,
          const int *lattice_bounds,
          const double *aspect_ratio)


The data members are
  1, 2 or 3 dimensions
  unsigned char _dimen;
  size of the array image in each direction
  int _bounds[3];
  real-valued size of the pixel or voxel
  double _aspect[3];
  inverses of the values of _aspect
  double inverse_aspect[3];
```

The default constructor **ImageBase()** can be used and the data members set with the functions

```
  virtual void setDimension(unsigned char dimension)
  virtual void setBounds(const int *lattice_bounds)
  virtual void setAspect(const double *aspect_ratio)
```

There also exist corresponding ‘get’ functions.

Constant pointers to the size and shape arrays are available using the functions

```
  const int* getBoundsArray() const
  const double* getAspectArray() const
```

The function

```
bool find_indices_nearest(const double* const coord,
                          int* const lattice_site,
                          double* const location)
```

finds the lattice sites nearest a given point. The input parameter **coord** uses the same unit of measure as the **aspect** variable of the lattice. The position is measured from one corner of the lattice. The output parameter **lattice_site** gives the lattice index and the output parameter **location** gives the distance from the center of the pixel or voxel that corresponds to the lattice index. The function returns 1 if the point is inside the lattice, zero otherwise.

The function

```
bool find_indices_nearest(const double* const coord,
                          int* const lattice_site,
                          double* const location,
                          const signed char* field_pos)
```

is the same as the previously defined function of the same name, except that the input variable **field_pos** defines the zero of the coordinate system: lower edge, middle, or upper edge. For example $field\_pos = (-1,-1,-1)$ corresponds to the default value of the previously defined function and $field\_pos = (0,0,0)$ corresponds to the center of the lattice. The function returns 1 if the point is inside the lattice, zero otherwise.

This class defines constants such as

```
static const int ImageBase::IMAGE_DATA_TYPE_INT = 6;
```

that are used by derived classes to define the type of the field.

toDataType The gobal functions **toDataType** in the namespace **BasicDataType** return an integer constant that gives the type of a field. The function

```
template <class U> int BasicDataType::toDataType()
```

is useful as a member of templated fields. The function

```
int BasicDataType::toDataType(const char* type_in)
```

returns a field type based on a character string that names the type.

Class Graph

- 1.1 Vector 11
  - 40 TNTVect 66

- 4 LimitRange 14

- 18 LinAlgScalar 36
  - 5 Number 15
    - 11 NumberTyped 24
  - 23 LinAlgScalarTyped 43

- 30 LinAlgVector 53
  - 58 LinAlgVectorSpace 90
    - 59 VecSpecificDim 92
  - 112 ImageField 208
    - 114 ImageFieldTyped 215

- **47** ReadOnlyNumArray 76
- **48** Timer 78
- **55** Vector1 86
- **56** Vector2 87
- **57** Vector3 88
- **75** BsplineEquations 112
- **76** Bspline 117
- **77** StencilHandle 121
- **86** ArbitrarySites 137
  - **93** ArbitraryMatrix 168

- **87** StencilSitesTag 145
- **88** StencilParams 147
  - **85** StencilSites 130
    - **92** StencilMatrix 164
- **89** TermsTag 152
- **90** PrecisionChoice 155
- **91** StencilTerms 156
- **103** ImageBase 182
  - **111** ImageFieldBase 205
    - **108** ImageFieldAssign 195
      - **114** ImageFieldTyped 215
- **105** ObjVar 188

106 LinTrans 190

107 MapDef 192

113 NewImageField 214

162 ImageFieldAlgorithms 274

163 FieldInterpolHelper 276

164 FieldInterpolAlgorithms 278

165 RegridBrick 280

166 GridSlice 282