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**  
   1.1 Vector — Basic TNT numerical vector.  

2. **CastToSelfType** — Contains a method to cast a base class to a derived class  
   2.1 cast_to_self_type — Casts a base class U to a derived class T.  

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

4. **LimitRange** — Converts between various primitive number types.  
   4.1 methods  

5. **Number** — Can represent any type of number.  
   5.1 cast underlying rep to Number pointer  
   5.1.1 usual methods  
   5.1.2 virtual methods  

6. **operator+**  

7. **operator-**  

8. **operator***  

9. **operator/**  

10. **GetNumber** — template parameterized getNumber()  
    10.1 getNumber — Gets a primitive number from wrapper class Number.  

11. **NumberTyped** — Contains one scalar of type T.  
    11.1 type definitions  
    11.1.1 operator- — A virtual function of class LinAlgScalar.  
    11.2 data  
    11.3 usual methods  
    11.4 methods  
    11.5 arithmetic methods  
    11.6 helper methods  
    11.7 virtual methods of LinAlgScalar  
    11.7.1 operator-  
    11.8 virtual methods of Number  

12. **operator+**  

13. **operator-**  

14. **operator***  

15. **operator/**  

16. **operator<<**  

17. **operator>>**  

18. **LinAlgScalar** — A scalar for linear algebra, independent of numeric type  
   18.1 data  
   18.1.1 lin_alg_scalar — Is non-zero only when the actual, highest level class is the base class  
   18.2 usual methods  
   18.2.1 "LinAlgScalar — Each instantiation has its own instantiation of a LinAlgScalar* lin_alg_scalar.so it is deleted when the object is deleted  
   18.3 virtual methods  
   18.4 public methods  

19. **operator+**  

---

This page was generated with the help of DDC++  
http://www.linuxsupportlive.com/doc++  
April 29, 2002
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>operator-</td>
<td>40</td>
</tr>
<tr>
<td>21</td>
<td>operator*</td>
<td>41</td>
</tr>
<tr>
<td>22</td>
<td>operator/</td>
<td>42</td>
</tr>
<tr>
<td>23</td>
<td>LinAlgScalarTyped — The class LinAlgScalarTyped&lt;T&gt; contains one scalar of type T</td>
<td>43</td>
</tr>
<tr>
<td>23.1</td>
<td>type definitions</td>
<td>44</td>
</tr>
<tr>
<td>23.2</td>
<td>data</td>
<td>46</td>
</tr>
<tr>
<td>23.3</td>
<td>methods</td>
<td>44</td>
</tr>
<tr>
<td>23.3.1</td>
<td>operator- — A virtual function of class LinAlgScalar</td>
<td>46</td>
</tr>
<tr>
<td>24</td>
<td>operator+</td>
<td>47</td>
</tr>
<tr>
<td>25</td>
<td>operator-</td>
<td>48</td>
</tr>
<tr>
<td>26</td>
<td>operator*</td>
<td>49</td>
</tr>
<tr>
<td>27</td>
<td>operator/</td>
<td>50</td>
</tr>
<tr>
<td>28</td>
<td>operator&lt;&lt;</td>
<td>51</td>
</tr>
<tr>
<td>29</td>
<td>operator&gt;&gt;</td>
<td>52</td>
</tr>
<tr>
<td>30</td>
<td>LinAlgVector — Base class for a vector used in linear algebra</td>
<td>53</td>
</tr>
<tr>
<td>30.1</td>
<td>data</td>
<td>56</td>
</tr>
<tr>
<td>30.1.1</td>
<td>lin_alg_vector — Is non-zero only when the actual, highest level class is the base class</td>
<td>56</td>
</tr>
<tr>
<td>30.2</td>
<td>usual methods</td>
<td>53</td>
</tr>
<tr>
<td>30.2.1</td>
<td>LinAlgVector — Each instantiation of LinAlgVector has its own lin_alg_vector_, created using clone().</td>
<td>54</td>
</tr>
<tr>
<td>30.3</td>
<td>virtual methods</td>
<td>54</td>
</tr>
<tr>
<td>30.4</td>
<td>public methods</td>
<td>55</td>
</tr>
<tr>
<td>31</td>
<td>operator+</td>
<td>57</td>
</tr>
<tr>
<td>32</td>
<td>operator-</td>
<td>58</td>
</tr>
<tr>
<td>33</td>
<td>operator*</td>
<td>59</td>
</tr>
<tr>
<td>34</td>
<td>operator+</td>
<td>60</td>
</tr>
<tr>
<td>35</td>
<td>operator-</td>
<td>61</td>
</tr>
<tr>
<td>36</td>
<td>operator*</td>
<td>62</td>
</tr>
<tr>
<td>37</td>
<td>operator+</td>
<td>63</td>
</tr>
<tr>
<td>38</td>
<td>operator-</td>
<td>64</td>
</tr>
<tr>
<td>39</td>
<td>operator*</td>
<td>65</td>
</tr>
<tr>
<td>40</td>
<td>TNTVect — A TNT vector with some modifications.</td>
<td>66</td>
</tr>
<tr>
<td>40.1</td>
<td>copy and assignment helper methods</td>
<td>69</td>
</tr>
<tr>
<td>40.1.1</td>
<td>convert — Copy of data members</td>
<td>69</td>
</tr>
<tr>
<td>40.2</td>
<td>fast copy and set</td>
<td>69</td>
</tr>
<tr>
<td>40.3</td>
<td>usual methods</td>
<td>67</td>
</tr>
<tr>
<td>40.4</td>
<td>methods of TNT Vector that return this</td>
<td>68</td>
</tr>
<tr>
<td>40.5</td>
<td>new methods not in TNT Vector</td>
<td>68</td>
</tr>
<tr>
<td>40.6</td>
<td>I/O</td>
<td>68</td>
</tr>
<tr>
<td>41</td>
<td>operator+ — TNTVect sum</td>
<td>70</td>
</tr>
<tr>
<td>42</td>
<td>operator- — TNTVect difference</td>
<td>71</td>
</tr>
<tr>
<td>43</td>
<td>operator* — TNTVect component by component product</td>
<td>72</td>
</tr>
<tr>
<td>44</td>
<td>dot_prod — TNTVect inner product</td>
<td>73</td>
</tr>
<tr>
<td>45</td>
<td>operator&lt;&lt; — TNTVect write to standard output.</td>
<td>74</td>
</tr>
<tr>
<td>46</td>
<td>operator&gt;&gt; — TNTVect read from standard input.</td>
<td>75</td>
</tr>
<tr>
<td>47</td>
<td>ReadOnlyNumArray — An array that can be declared Read Only</td>
<td>76</td>
</tr>
<tr>
<td>47.1</td>
<td>data</td>
<td>77</td>
</tr>
<tr>
<td>47.2</td>
<td>usual methods</td>
<td>76</td>
</tr>
<tr>
<td>47.3</td>
<td>methods</td>
<td>77</td>
</tr>
<tr>
<td>Page</td>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>48</td>
<td>Timer — <em>A stopwatch</em></td>
<td>78</td>
</tr>
<tr>
<td>48.1</td>
<td>data</td>
<td>79</td>
</tr>
<tr>
<td>48.2</td>
<td>public typedef and data</td>
<td>78</td>
</tr>
<tr>
<td>48.3</td>
<td>usual methods</td>
<td>78</td>
</tr>
<tr>
<td>48.4</td>
<td>methods</td>
<td>79</td>
</tr>
<tr>
<td>49</td>
<td>operator&lt;&lt;</td>
<td>80</td>
</tr>
<tr>
<td>50</td>
<td>operator&gt;&gt;</td>
<td>81</td>
</tr>
<tr>
<td>51</td>
<td>operator&lt;&lt;</td>
<td>82</td>
</tr>
<tr>
<td>52</td>
<td>operator&gt;&gt;</td>
<td>83</td>
</tr>
<tr>
<td>53</td>
<td>operator&lt;&lt;</td>
<td>84</td>
</tr>
<tr>
<td>54</td>
<td>operator&gt;&gt;</td>
<td>85</td>
</tr>
<tr>
<td>55</td>
<td>Vector1 — <em>Include access by functions used by Vertex in order to generalize algorithms</em></td>
<td>86</td>
</tr>
<tr>
<td>55.1</td>
<td>public methods</td>
<td>86</td>
</tr>
<tr>
<td>56</td>
<td>Vector2 — <em>Include access by functions used by Vertex in order to generalize algorithms</em></td>
<td>87</td>
</tr>
<tr>
<td>56.1</td>
<td>public methods</td>
<td>87</td>
</tr>
<tr>
<td>57</td>
<td>Vector3 — <em>Include access by functions used by Vertex in order to generalize algorithms</em></td>
<td>88</td>
</tr>
<tr>
<td>57.1</td>
<td>public methods</td>
<td>88</td>
</tr>
<tr>
<td>58</td>
<td>LinAlgVectorSpace</td>
<td>90</td>
</tr>
<tr>
<td>58.1</td>
<td>casting to actual type</td>
<td>91</td>
</tr>
<tr>
<td>58.2</td>
<td>usual methods</td>
<td>90</td>
</tr>
<tr>
<td>58.3</td>
<td>virtual methods</td>
<td>90</td>
</tr>
<tr>
<td>59</td>
<td>VecSpecificDim — <em>T can be Vector1, Vector2, or Vector3</em></td>
<td>92</td>
</tr>
<tr>
<td>59.1</td>
<td>data</td>
<td>96</td>
</tr>
<tr>
<td>59.2</td>
<td>usual methods</td>
<td>93</td>
</tr>
<tr>
<td>59.2.1</td>
<td>VecSpecificDim — <em>default constructor</em></td>
<td>93</td>
</tr>
<tr>
<td>59.3</td>
<td>static methods</td>
<td>93</td>
</tr>
<tr>
<td>59.4</td>
<td>access to data</td>
<td>94</td>
</tr>
<tr>
<td>59.5</td>
<td>operators</td>
<td>94</td>
</tr>
<tr>
<td>59.6</td>
<td>virtual functions of LinAlgVector</td>
<td>94</td>
</tr>
<tr>
<td>59.7</td>
<td>virtual functions of LinAlgVectorSpace</td>
<td>95</td>
</tr>
<tr>
<td>59.8</td>
<td>linear algebra functions</td>
<td>95</td>
</tr>
<tr>
<td>59.9</td>
<td>virtual functions of LinAlgVector</td>
<td>96</td>
</tr>
<tr>
<td>60</td>
<td>operator+</td>
<td>97</td>
</tr>
<tr>
<td>61</td>
<td>operator-</td>
<td>98</td>
</tr>
<tr>
<td>62</td>
<td>operator*</td>
<td>99</td>
</tr>
<tr>
<td>63</td>
<td>operator+</td>
<td>100</td>
</tr>
<tr>
<td>64</td>
<td>operator-</td>
<td>101</td>
</tr>
<tr>
<td>65</td>
<td>operator*</td>
<td>102</td>
</tr>
<tr>
<td>66</td>
<td>operator+</td>
<td>103</td>
</tr>
<tr>
<td>67</td>
<td>operator-</td>
<td>104</td>
</tr>
<tr>
<td>68</td>
<td>operator*</td>
<td>105</td>
</tr>
<tr>
<td>69</td>
<td>operator&lt;&lt; &lt; Vector1&gt;</td>
<td>106</td>
</tr>
<tr>
<td>70</td>
<td>operator&gt;&gt; &lt; Vector1&gt;</td>
<td>107</td>
</tr>
<tr>
<td>71</td>
<td>operator&lt;&lt; &lt; Vector2&gt;</td>
<td>108</td>
</tr>
<tr>
<td>72</td>
<td>operator&gt;&gt; &lt; Vector2&gt;</td>
<td>109</td>
</tr>
<tr>
<td>73</td>
<td>operator&lt;&lt; &lt; Vector3&gt;</td>
<td>110</td>
</tr>
<tr>
<td>74</td>
<td>operator&gt;&gt; &lt; Vector3&gt;</td>
<td>111</td>
</tr>
<tr>
<td>75</td>
<td>BsplineEquations — <em>B splines for interpolation.</em></td>
<td>112</td>
</tr>
<tr>
<td>75.1</td>
<td>basic equations</td>
<td>112</td>
</tr>
</tbody>
</table>
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 bsplinepair3 — input x is centered between two peaks .................................. 115
75.1.5 bsplinepair3_derivative — input x is centered between two peaks ................. 115
75.1.6 bsplinepair3_integral — input x is centered between two peaks .................... 115
75.1.7 bsplinepair3_derivative — 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 .................................. 116
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 ............................................................................................................. 117
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
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 convertTree — Call convertTree 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 makePointTerms — A function to generate polynomial terms ................. 159
    91.7.2 makeRngTerms — A function to generate terms averaged over a box ....... 160
    91.7.3 makeGradientTerms — 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 fillMatrix — Fill matrix based on field for a specific stencil. ................. 167
  92.4 usual methods .................................................................................................. 164
  92.5 data .................................................................................................................... 165
    92.5.1 colByColMatrix — matrix of polynomial terms and sites. ..................... 165
  92.6 methods. ......................................................................................................... 165
    92.6.1 fillMatrix — 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 colByColMatrix — 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
| 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_interp — 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 |
namespace TNT

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

**Names**

1.1 template<class T> class Vector

*Basic TNT numerical vector.*

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

**Inheritance**

1.1 Vector

Basic TNT numerical vector.

Template Numerical Toolkit (TNT): Linear Algebra Module

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

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 instantiation 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
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

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

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)
class **Number** : virtual public LinAlgScalar

*Can represent any type of number.*

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

### Inheritance

```
                           18
                          LinAlgScalar  
                           5
                             Number
                           11
                             NumberTyped
```

### 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 instantiation has its own instantiation 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

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
5.1.3

virtual Number& \texttt{operator=} (const Number& object\_in)

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

assignment

5.1.4

\texttt{~Number ()}

destructor

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

5.1.2

\textbf{virtual methods}

\begin{itemize}
  \item virtual Number* \texttt{newNumber () const}
  \item virtual Number* \texttt{cloneNumber () const}
  \item virtual char \texttt{getNumber\_char () const}
  \item virtual signed char \texttt{getNumber\_signed\_char () const}
  \item virtual unsigned char \texttt{getNumber\_unsigned\_char () const}
  \item virtual short \texttt{getNumber\_short () const}
  \item virtual unsigned short \texttt{getNumber\_unsigned\_short () const}
  \item virtual int \texttt{getNumber\_int () const}
  \item virtual unsigned int \texttt{getNumber\_unsigned\_int () const}
  \item virtual long \texttt{getNumber\_long () const}
  \item virtual unsigned long \texttt{getNumber\_unsigned\_long () const}
  \item virtual float \texttt{getNumber\_float () const}
\end{itemize}
virtual double getNumber() const
virtual long double getNumber() 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)
inline  Number \texttt{operator+} (const Number& A, const Number& B)

In file ../LinAlg/number.hh:963
inline  Number operator- (const Number& A, const Number& B)

In file ../LinAlg/number.hh:967
inline Number operator* (const Number& A, const Number& B)

In file ../LinAlg/number.hh:971
inline  Number operator/ (const Number& A, const Number& B)

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

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

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&
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
template<class T> inline NumberTyped<T> operator+ (const NumberTyped<T>& a, const NumberTyped<T>& b)
template<class T> inline NumberTyped<T> operator- (const NumberTyped<T>& a, const NumberTyped<T>& b)
template<class T>inline NumberTyped<T> operator* (const NumberTyped<T> & a, const NumberTyped<T> & b)
template<class T> inline NumberTyped<T> operator/ (const NumberTyped<T>& a, const NumberTyped<T>& b)
std::ostream& operator<< (std::ostream& s, const LinAlgScalar& r)

In file ../LinAlg/linalg_vector.hh:1443
std::istream & operator >> (std::istream & s, LinAlgScalar & r)

In file ../LinAlg/linalgvector.hh:1445
class LinAlgScalar

A scalar for linear algebra, independent of numeric type

In file ../LinAlg/lin\_alg\_vector.hh:1449

Inheritance

LinAlgScalar

5

Number

23

LinAlgScalarTyped

Public Members

18.2 usual methods

18.3 virtual methods

18.4 public methods

Protected Members

18.1 data

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 instantiation has its own instantiation of a LinAlgScalar* lin_{alg_scalar} so it is deleted when the object is deleted

In file ../LinAlg/lin_{alg_vector}.hh:1487

Each instantiation has its own instantiation 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)
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

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
inline LinAlgScalar \texttt{operator+} (const LinAlgScalar& A, const LinAlgScalar& B)

In file ../LinAlg/linalgvector.hh:1580
inline LinAlgScalar operator- (const LinAlgScalar& A, const LinAlgScalar& B)

In file ../LinAlg/lin_vector.hh:1584
inline LinAlgScalar operator* (const LinAlgScalar& A, const LinAlgScalar& B)

In file ../LinAlg/lin_vector.hh:1588
inline LinAlgScalar operator/ (const LinAlgScalar& A, const LinAlgScalar& B)

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

LinAlgScalar

LinAlgScalarTyped

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> *
close() 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.

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/linalgvector.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_
template<class T> inline LinAlgScalarTyped<T> operator+ (const LinAlgScalarTyped<T> & a, const LinAlgScalarTyped<T> & b)
template<class T> inline LinAlgScalarTyped<T> operator- (const LinAlgScalarTyped<T> & a, const LinAlgScalarTyped<T> & b)
template<class T> inline LinAlgScalarTyped<T> operator* (const LinAlgScalarTyped<T> & a, const LinAlgScalarTyped<T> & b)
template<class T> inline LinAlgScalarTyped<T> operator/ (const LinAlgScalarTyped<T> & a, const LinAlgScalarTyped<T> & b)

In file ../LinAlg/linear_vector.hh:1822
std::ostream& operator<<(std::ostream& s, const LinAlgVector& r)

In file ../LinAlg/lin_alg_vector.hh:1828
std::istream& operator>>(std::istream& s, LinAlgVector& r)

In file ../LinAlg/linAlgVector.hh:1830
class LinAlgVector

Base class for a vector used in linear algebra

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

Inheritance

LinAlgVector
    LinAlgVectorSpace
    ImageField

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
30

LinAlgVector

Names

LinAlgVector()

30.2.1

LinAlgVector (const LinAlgVector& lav)

Each instantiation of LinAlgVector has its own lin\_alg\_vector, created using clone().

virtual LinAlgVector&
    operator= (const LinAlgVector& lav)

virtual ~LinAlgVector()

In file ../LinAlg/lin\_alg\_vector.hh:1863

Each instantiation 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

define

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)
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

**Names**

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

In file ../LinAlg/lin_alg_vector.hh:2047
inline LinAlgVector operator* (const LinAlgScalar& las, const LinAlgVector& B)

In file ../LinAlg/linalg_vector.hh:2051
inline LinAlgVector \texttt{operator+} (const double& D, const LinAlgVector& B)
inline LinAlgVector \texttt{operator-} (const double& D, const LinAlgVector& B)

In file ../LinAlg/lin\_alg\_vector.hh:2059
inline  LinAlgVector operator* (const double& D, const LinAlgVector& B)

In file ../LinAlg/lin_alg_vector.hh:2063
inline LinAlgVector operator+ (const LinAlgVector& A, const LinAlgVector& B)

In file ../LinAlg/linalgvector.hh:2067
inline LinAlgVector \textbf{operator-} (const LinAlgVector& A, const LinAlgVector& B)

In file ../LinAlg/linalgvector.hh:2071
inline LinAlgVector operator*(const LinAlgVector& A, const LinAlgVector& B)

In file ../LinAlg/lin_vector.hh:2075
template<class T> class TNTVec : public Vector<T>

A TNT vector with some modifications.

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

Inheritance

1.1 Vector

TNTVec

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

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 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)

April 29, 2002 69
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.
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)

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.
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.
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.
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.
template<class T> class ReadOnlyNumArray

An array that can be declared Read Only

In file ../LinAlg/readonly_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 TNTVec<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 setValue (int num_elem, const T* v)

47.1 data

Names

TNTVct<T> _v
bool _ro
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::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`
ostream& operator<< (ostream& s, const Vector1& A)
istream& operator>>(istream& s, Vector1& A)

In file ../LinAlg/vector123.hh:26
ostream& operator<< (ostream& s, const Vector2& A)
istream& \texttt{operator>>} (istream& s, Vector2& A)

In file ../LinAlg/vector123.hh:2668
ostream& operator<< (ostream& s, const Vector3& A)

In file ../LinAlg/vector123.hh:2670
istream& operator>>(istream& s, Vector3& A)

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

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
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
class LinAlgVectorSpace : virtual public LinAlgVector

In file ../LinAlg/vector123.hh:28

Inheritance

LinAlgVectorSpace
  ^
  |
  V
LinAlgVector

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
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  

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

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

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

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

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

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_
template<class T> inline LinAlgVector \texttt{operator+} (const double& A, const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3221
template<class T> inline LinAlgVector operator- (const double& A, const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3228
template<class T> inline LinAlgVector operator* (const double& A, const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3236
template<class T> inline LinAlgVector operator+ (const LinAlgScalar& las, const VecSpecificDim<T>& B)
template<class T> inline LinAlgVector operator- (const LinAlgScalar& las,
   const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3248
template<class T> inline LinAlgVector operator* (const LinAlgScalar& las, const VecSpecificDim<T>& B)
template<class T> inline LinAlgVector operator+ (const VecSpecificDim<T>& A, const VecSpecificDim<T>& B)

In file ../LinAlg/vector123.hh:3258
template<class T>inline LinAlgVector operator- (const VecSpecificDim<T>& A, const VecSpecificDim<T>& B)
template<class T>inline LinAlgVector operator* (const VecSpecificDim<T>& A, const VecSpecificDim<T>& B)
template<> std::ostream& operator<< <Vector1> (std::ostream& s, const Vector1& A)

In file ../LinAlg/vector123.hh:3279
template<> std::istream& operator>> <Vector1> (std::istream& s, VecSpecificDim<Vector1>& A)

In file ../LinAlg/vector123.hh:323

April 29, 2002 107
template<> std::ostream& operator<< <Vector2> (std::ostream& s, const Vector2& A)
template<> std::istream& operator>> <Vector2> (std::istream& s, VecSpecificDim<Vector2>& A)

In file ../LinAlg/vector123.hh:3291
template<> std::ostream& operator<< <Vector3> (std::ostream& s, const Vector3& A)

In file ../LinAlg/vector123.hh:3295
template<> std::istream& operator>> <Vector3> (std::istream& s, VecSpecificDim<Vector3>& A)

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

B splines for interpolation.

In file ./Basic/bspline.hh:3336

Public Members

75.1 basic equations .............................................. 112

B splines for interpolation.

\[
B^n(x) = \sum_{k=0}^{n+1} \frac{(-1)^k(n+1)\left(\frac{n+1}{2} + x - k\right)^n}{(n+1-k)!k!}
\]

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

Author:  Alan Louis Scheinine

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)

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)

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

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

75.1.9  static inline double  
    bspline4 (double x)

75.1.10 static inline double  
    bspline4_derivative (double x)

75.1.11 static inline double  
    bspline4_integral (double x, double y)

75.1.12 static inline double  
    bspline4_integral (double x)

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

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

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

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

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

```c
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

```c
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

```c
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

```c
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
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

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)

April 29, 2002 117
virtual ~Bspline ()

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 double 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
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]
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

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 T*  operator T* () const
inline 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_
extern const int POINT_VALUE_MODE

Value set in stencil_matrix.C

In file ../Basic/interpol.hh:4226
extern const int BOX_VALUE_MODE

Value set in stencil_matrix.C

In file ../Basic/interpol.hh:4228
extern const int PRECISION_LEVEL1

Value set in Field/field_interpol_algorithms.C

In file ../Basic/interpol.hh:4231
extern const int PRECISION_LEVEL2

Value set in Field/field_interpol_algorithms.C

In file ../Basic/interpol.hh:4233
extern const int PRECISION_LEVEL3

Value set in Field/field_interpolationalgorithms.C

In file ../Basic/interpol.hh:4235
extern const int PRECISION_LEVEL4

Value set in Field/field_interpol_algorithms.C

In file ../Basic/interpol.hh:4237
LinAlg read_only_num_array.hh const int MAX_STENCIL_SITES

In file ../Basic/stencil_sites.hh:4291
class StencilSites : public StencilParams

A stencil of sites.

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

Inheritance

StencilParams

StencilSites

StencilMatrix

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

StencilSites (int tag, in)
   Sets the x, y, [z] sites of a stencil.

StencilSites (const StencilSites& 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>
\texttt{stencil\_sites\_x} \hspace{1cm} \textit{X positions of the stencil sites} \hspace{1cm} 132

85.8.2  ReadOnlyNumArray<signed char>
\texttt{stencil\_sites\_y} \hspace{1cm} \textit{Y positions of the stencil sites} \hspace{1cm} 132

85.8.3  ReadOnlyNumArray<signed char>
\texttt{stencil\_sites\_z} \hspace{1cm} \textit{Z positions of the stencil sites} \hspace{1cm} 132

---

85.8.1

\textbf{ReadOnlyNumArray<signed char> \texttt{stencil\_sites\_x}}

\textit{X positions of the stencil sites}

In file \texttt{./Basic/stencil\_sites.hh:4606}

\textit{X positions of the stencil sites}

---

85.8.2

\textbf{ReadOnlyNumArray<signed char> \texttt{stencil\_sites\_y}}

\textit{Y positions of the stencil sites}

In file \texttt{./Basic/stencil\_sites.hh:4611}

\textit{Y positions of the stencil sites}

---

85.8.3

\textbf{ReadOnlyNumArray<signed char> \texttt{stencil\_sites\_z}}

\textit{Z positions of the stencil sites}

In file \texttt{./Basic/stencil\_sites.hh:4616}

\textit{Z positions of the stencil sites}
static methods

Names

static int get_extent (int tag_in)

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]
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.
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 () \hspace{1em} Default constructor.

86.4.1 ArbitrarySites (int size_in, const double* stencil_sites_xx)
\hspace{1em} Constructor. ........................................... 138

86.4.2 ArbitrarySites (int size_in, const double* stencil_sites_xx,
\hspace{1em} const double* stencil_sites_yy)
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.
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
86.5.2 ReadOnlyNumArray<double>
   stencil_sites_y
   Y positions of the sites
86.5.3 ReadOnlyNumArray<double>
   stencil_sites_z
   Z positions of the sites

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

```cpp
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

```cpp
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)
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


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

```cpp
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**

```cpp
int _tag
```

*Integer-valued tag*
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

<table>
<thead>
<tr>
<th>88.6.1</th>
<th>inline</th>
<th>StencilSitesTag</th>
<th>getTag () const</th>
<th>Get tag value.</th>
<th>149</th>
</tr>
</thead>
<tbody>
<tr>
<td>88.6.2</td>
<td>inline</td>
<td>int</td>
<td>getSize () const</td>
<td>Get total size.</td>
<td>149</td>
</tr>
<tr>
<td>88.6.3</td>
<td>inline</td>
<td>int</td>
<td>getDimension () const</td>
<td>Get number of dimensions (2 or 3).</td>
<td>149</td>
</tr>
</tbody>
</table>
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

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:5046

Copy of data members

88.1.1

inline  void convert (const StencilParams& object_in)

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
int  _size
int  _dimen

Indicates the stencil choice.
The size of the array of sites.
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.

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
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

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**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TermsTag ()</td>
<td>default constructor</td>
</tr>
<tr>
<td>TermsTag (int tag_in, int basis_in)</td>
<td>constructor</td>
</tr>
<tr>
<td>TermsTag (const TermsTag&amp; object_in)</td>
<td>copy constructor</td>
</tr>
<tr>
<td>TermsTag&amp; operator= (const TermsTag&amp; object_in)</td>
<td>assignment</td>
</tr>
<tr>
<td>~TermsTag ()</td>
<td>destructor</td>
</tr>
</tbody>
</table>

### 89.2.2 data access methods

**Names**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inline int getSize() const</td>
<td>Get total size. ............................ 153</td>
</tr>
<tr>
<td>inline int getSize() const</td>
<td>Get number of dimensions. ............ 154</td>
</tr>
</tbody>
</table>

#### 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 \textbf{getDimension} () const

Get number of dimensions.

In file "/Basic/stencil\_terms.hh:5316

Get number of dimensions.

\textbf{Return Value:}\quad dimension

89.2.3

\textbf{static methods}

Names

static bool \textbf{check\_terms\_tag} (int tag\_in, int basis\_in)
class PrecisionChoice

Holds choice of stencil and choice of basis for interpolation

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

Public Members

90.1 data

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
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

91.4 Constructors

Names
91.4.1 StencilTerms () \textit{Default constructor.} ......................... 157
StencilTerms (int tag\_in) \textit{constructor}
StencilTerms (int tag\_in, int basis\_in) \textit{constructor}
StencilTerms (const TermsTag& tag\_in)
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.
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
StencilVector \texttt{zterms}

The \textit{z} values of an array of vectors for the gradient

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

The \textit{z} values of an array of vectors for the gradient

\textbf{methods}

\textbf{Names}

\begin{itemize}
  \item \texttt{make\_point\_terms} (const double *\texttt{location})
  \textit{A function to generate polynomial terms} \ldots 159
  \item \texttt{make\_ntgr\_terms} (const double *\texttt{box})
  \textit{A function to generate terms averaged over a box} \ldots 160
  \item \texttt{make\_gradient\_terms} (const double *\texttt{location})
  \textit{A function to generate gradient vector terms} 160
  \item \texttt{getTag} () const  \textit{Get tag value.} 160
  \item \texttt{getSize} () const  \textit{Get total size.} 160
  \item \texttt{getDimension} () const  \textit{Get number of dimensions.} 161
\end{itemize}

\textbf{91.7.1}

\texttt{void make\_point\_terms} (const double *\texttt{location})

\textit{A function to generate polynomial terms}

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

A function to generate polynomial terms
### 91.7.2

```c
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

```c
void makeGradient_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

```c
inline TermsTag getTag () const
```

*Get tag value.*

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

Get tag value.

**Return Value:**

*tag*

### 91.7.5

```c
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

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)
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

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
**92.3**

**methods**

**Names**

<table>
<thead>
<tr>
<th>TermsTag</th>
<th>_terms_tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>int num_rows</td>
<td>_num_rows</td>
</tr>
<tr>
<td>int num_cols</td>
<td></td>
</tr>
</tbody>
</table>

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

*Fill matrix based on field for a specific stencil.*

```
fill_matrix (int szize, const double* position, const double* aspect, int value_mode)
```

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

Fill matrix based on field for a specific stencil.

**Parameters:**

- **positionCan**
  - use a different interpolation function around each pixel/voxel, hence, the position can be zero (0,0,0,0[0,0]).
- **field**
  - image values
- **value_mode**
  - BOX_VALUE_MODE implies matrix terms are the value at the center point. BOX_VALUE_MODE implies matrix terms are an average over the pixel/voxel.
class ArbitraryMatrix : public ArbitrarySites

A matrix that relates polynomial coefficients to field values.

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

Inheritance

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

Names

ArbitraryMatrix () Default constructor.
ArbitraryMatrix (ArbitrarySites& object, int tag, int basis)
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.  

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.  

93.6.2 inline int numTerms () const Get number of terms.  

93.6.3 inline int numSites () const Get number of sites.  

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:**

\[ \text{tag} \]

### 93.6.2

**inline int numTerms () const**

Get number of terms.

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

Get number of terms.

**Return Value:**

\[ \text{number of terms} \]

### 93.6.3

**inline int numSites () const**

Get number of sites.

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

Get number of sites.

**Return Value:**

\[ \text{number of sites} \]

### 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 ()
typedef TNTVec<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 alann Exp $
typedef TNTVect<double> * StencilVector_pointer

In file ../Basic/stencil_vector.hh:6906
typedef TNTVector<double> & StencilVector_ref
typedef const TNTVector<double> * StencilVector_const_pointer
typedef const TNTVect<double> & StencilVector_const_ref

In file ../Basic/stencil_vector.hh:6912
typedef TNT::Vector<double> iterator StencilVector_iterator

In file../Basic/stencil_vector.hh:6914
typedef TNT::Vector<double> const_iterator StencilVector_const_iterator
StencilVector_pointer **newStencilVector** ()

In file ../Basic/stencil_vector.hh:6918
StencilVector::pointer newStencilVector (int n)

In file ../Basic/stencil_vector.hh:6920
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 .......................................................... 183

Protected Members

103.1 copy and assignment helper methods .......................... 185
103.2 data .............................................................. 186
103.3 methods .......................................................... 186

A few words should be said about \texttt{aspect} and \texttt{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 \texttt{aspect} array be considered a constant that indicates the physical size because in this way two different fields can be compared.

The array \texttt{inverse\_aspect} is a duplication of information, it is simply the inverse of the \texttt{aspect} array. It is generated because the inverse values are used often in a particular calculation.

\textbf{Author:} Alan Louis Scheinine

\textbf{Version:} \$Id: image_base.hh,v 1.2 2002/03/16 17:09:31 alan Exp \$

April 29, 2002 182
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
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.

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**
- `void convert (const ImageBase& object_in)`

*Copy of data members* .......... 185

- `void convert_tree (const ImageBase& object_in)`

*Call convert_tree on each parent class then call convert* .......... 186

---

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

*Copy of data members*

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

Copy of data members
103.1.2

```cpp
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

April 29, 2002
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 toDataT ype ()  Conversion from a type an integer descriptor. ............................................ 187
104.2 int toDataT ype (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 toDataT ype ()

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 toDataT ype (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
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_
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

106.1

data

Names


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.
class MapDef

Linear mapping between two fields.

In file ../Field/mapdef.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**

<table>
<thead>
<tr>
<th>107.1.1</th>
<th>LinTrans</th>
<th><strong>linear</strong></th>
<th>a linear transformation</th>
<th>193</th>
</tr>
</thead>
<tbody>
<tr>
<td>107.1.2</td>
<td>signed char</td>
<td><strong>beg_field_pos</strong> [3]</td>
<td>left justify, center, or right justify initial field</td>
<td>193</td>
</tr>
<tr>
<td>107.1.3</td>
<td>signed char</td>
<td><strong>end_field_pos</strong> [3]</td>
<td>left justify, center, or right justify destination field</td>
<td>194</td>
</tr>
</tbody>
</table>

**107.1.1**

**LinTrans linear**

*a linear transformation*

In file ../Field/map_def.hh:7505

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.
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 instantiation 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 TNTVec<T>& () const
inline operator TNTVec<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)
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)
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+= (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//= (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)

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)

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)
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.
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.
template<class T> class ImageFieldBase : public ImageBase

Contains an array of field values.

In file ../Field/image_field_base.hh:8380

Inheritance

ImageBase

ImageFieldBase

ImageFieldAssign

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: $Id: image_field_base.hh,v 1.5 2002/04/10 21:00:43 alan Exp$

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)
class **ImageField** : virtual public **LinAlgVector**

Abstract class that declares image methods.

In file ../Field/image_field.hh:8534

**Inheritance**

```
LinAlgVector
    
ImageField
    
ImageFieldTyped
```

**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 makeStencil_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
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, int iy, unsigned int v)
virtual void setImage (int ix, int iy, int iz, unsigned int v)
virtual void setImage (int ix, int iy, int iz, long v)
virtual void setImage (int ix, int iy, int iz, long double 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, int iy, int iz, float v)
virtual void setImage (int iy, int iy, int iz, float v)
virtual void setImage (int iy, int iy, int iz, double v)
virtual void setImage (int iy, int iy, int iz, long double v)
virtual void setImage (int iy, unsigned long v)
virtual void setImage (int iy, int iy, unsigned long v)
virtual void setImage (int iy, int iy, int iz, long double v)
virtual ImageField* new_project_x (int lower, int upper, int mode)
virtual ImageField* new_project_y (int lower, int upper, int mode)
virtual ImageField* new_project_z (int lower, int upper, int mode)
virtual ImageField* new_project_x_avg (int lower, int upper)
virtual ImageField* new_project_y_avg (int lower, int upper)
virtual ImageField* new_project_z_avg (int lower, int upper)
virtual ImageField* new_project_x_max (int lower, int upper)
virtual ImageField* new_project_y_max (int lower, int upper)
virtual ImageField* new_project_z_max (int lower, int upper)
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 \( q \),

\[
I_i = I(p_i) = c_j f_j(p_i) = c_j F_{ji}.
\]

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 \( c_i^q \) can be calculated using the inverse of \( F \),

\[
I_i = (F^{-1})_{ji} = c_i^q.
\]

If the information available is not the intensity at a point, but rather, the intensity averaged over a box \( b \), then we can write

\[
\langle I^q \rangle b_i = c_j^q \langle f_j \rangle b_i \equiv c_j^q G_{ji}.
\]

The coefficients \( c_i^q \) 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 |

April 29, 2002 213
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

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)
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**

```
LinAlgVector -> ImageField -> ImageBase -> ImageFieldBase -> ImageFieldAssign
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>103</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>111</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>108</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>114</td>
</tr>
</tbody>
</table>

LinAlgVector

ImageField

ImageBase

ImageFieldBase

ImageFieldAssign

ImageFieldTyped
```

**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

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<
 & 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`

virtual

**~ImageFieldTyped** ()  `destructor`

### 114.5

**methods**

**Names**

```cpp
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**

```cpp
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 ()
```
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.

---

**virtual functions of LinAlgVector**

**Names**

inline LinAlgVector&
operator= (const LinAlgScalar& las)
inline LinAlgVector&
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  getBounds_array () const
get Aspect_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, 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 shortgetImage_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
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)
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, int iy, int iz) U v)
template<class U> void
setImage_primitive (int iy, int iy, int iz) U v)
template<class U> void
setImage_primitive (int ix, int iy, int iz, U v)

114.3
copy and assignment helper methods
114 ImageFieldTyped

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 spect (int i) const
inline int dmsn () const
inline const int* bnds_array () const
inline const double* spect_array () const
template<class T> inline ImageFieldTyped<T> operator+ (const Number& A, const ImageFieldTyped<T>& B)
template<class T>inline  ImageFieldTyped<T>  operator- (const Number& A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typedef.hh:10125
template<class T> inline ImageFieldTyped<T> operator* (const Number& A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator+ (char A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_TYPED.hh:10135
template<class T> inline ImageFieldTyped<T> operator- (char A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_TYPED.hh:10140
template<class T> inline ImageFieldTyped<T> operator* (char A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_TYPED.hh:10145
template<class T>inline ImageFieldTyped<T> operator+ (signed char A, const ImageFieldTyped<T>& B)
template<class T>inline ImageFieldTyped<T> operator- (signed char A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10155
template<class T> inline ImageFieldTyped<T> operator* (signed char A, const ImageFieldTyped<T>& B)

In file ../Field/image_fieldTyped.hh:10160
template<class T> inline ImageFieldTyped<T> operator+ (unsigned char A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator- (unsigned char A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator* (unsigned char A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator+ (short A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_TYPED.hh:10180
template<class T> inline ImageFieldTyped<T> operator- (short A, const ImageFieldTyped<T>& B)
template<class T>inline ImageFieldTyped<T> operator* (short A, const ImageFieldTyped<T>& B)

In file ../Field/image_field Typed.hh:10190
template<
    class T>
    inline ImageFieldTyped<T> operator+ (unsigned short A,
    const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator- (unsigned short A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator* (unsigned short A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_TYPED.hh:10205
template<class T>inline ImageFieldTyped<T> operator+ (int A, const ImageFieldTyped<T>& B)

In file ../Field/image_fieldTyped.hh:10210
template<class T> inline ImageFieldTyped<T> operator- (int A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_TYPED.hh:10215
template<class T> inline ImageFieldTyped<T> operator* (int A, const ImageFieldTyped<T>& B)

In file ../Field/image_fieldTyped.hh:10220
```cpp
template<class T> inline ImageFieldTyped<T> operator+ (unsigned int A, const ImageFieldTyped<T>& B)
```

In file `../Field/image_field_typed.hh:10225`
template<class T>inline ImageFieldTyped<T> operator- (unsigned int A, const ImageFieldTyped<T>& B)

In file ../../../Field/image_fieldTyped.hh:10230
template<class T>inline ImageFieldTyped<T> operator* (unsigned int A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10235
template<class T> inline ImageFieldTyped<T> operator+ (long A, const ImageFieldTyped<T>& B)
template<class T>inline ImageFieldTyped<T> operator-(long A, const ImageFieldTyped<T>& B)

In file ../Field/image_field Typed.hh:10245
template<class T>inline ImageFieldTyped<T> operator* (long A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator+ (unsigned long A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator- (unsigned long A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator* (unsigned long A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator+ (float A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator- (float A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_TYPED.hh:10275
template<class T> inline ImageFieldTyped<T> operator* (float A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator+ (double A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_typed.hh:10285
template<
 class T>
 inline
 ImageFieldTyped<T> operator- (double A, const ImageFieldTyped<T> & B)

In file ../Field/image_field_TYPED.hh:10290
template<class T> inline ImageFieldTyped<T> operator* (double A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator+ (long double A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator- (long double A, const ImageFieldTyped<T>& B)

In file ../Field/image_field_TYPED.hh:10305
template<class T> inline ImageFieldTyped<T> operator* (long double A, const ImageFieldTyped<T>& B)
template<class T>inline ImageFieldTyped<T> operator+ (const LinAlgScalar& las, const ImageFieldTyped<T>& B)
template<class T>inline ImageFieldTyped<T> operator- (const LinAlgScalar& las, const ImageFieldTyped<T>& B)

In file ../Field/image_field_TYPED.hh:10320
template<class T>inline ImageFieldTyped<T> operator* (const LinAlgScalar& las, const ImageFieldTyped<T>& B)

In file ../Field/image_field.Typed.hh:10325
template<class T> inline ImageFieldTyped<T> operator+ (const ImageFieldTyped<T>& A, const ImageFieldTyped<T>& B)
template<class T>inline ImageFieldTyped<T> operator- (const ImageFieldTyped<T>& A, const ImageFieldTyped<T>& B)
template<class T> inline ImageFieldTyped<T> operator* (const ImageFieldTyped<T>& A, const ImageFieldTyped<T>& B)
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.
template<class T> std::istream& operator>>(std::istream &s, ImageFieldTyped<T> &A)

ImageFieldTyped read from standard input.

In file ../Field/image_field typings.hh:10355

ImageFieldTyped read from standard input.
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)
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)
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 $

usual methods

Names

FieldInterpolHelper () default constructor

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* \texttt{nvrs}
StencilVector \texttt{rhs}
StencilVector \texttt{coefs}
StencilTerms* \texttt{stencil}

163.3 initialization

Names
\texttt{void set_defaults()}

April 29, 2002 277
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

```c++
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

```cpp
ImageBase(int dimension, int lattice_bounds[3], double aspect_ratio[3])
```

are mapped onto fldntrpl.

164.2

private static methods

Names

```c++
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 makeStencilRhsToo (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)
```
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:

<table>
<thead>
<tr>
<th>stencil</th>
<th>basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>

Basis 1 is Taylor expansion. Basis 2 are Gaussians. Basis 9 gives bad results and is included only for testing.

Author: Alan Louis Scheinine


April 29, 2002 280
165.1 usual methods

Names

RegridBrick () \textit{default constructor}

RegridBrick (const RegridBrick& object\_in) \\
\textit{copy constructor}

RegridBrick& \textbf{operator=} (const RegridBrick& object\_in) \\
\textit{assignment}

~RegridBrick () \textit{destructor}

165.2 static methods

Names

static ImageField* \\
\textbf{new\_brick\_from\_aspect} (int image\_data\_type, \\
const ImageField& fld\_in, \\
const double* aspect\_ratio, \\
const PrecisionChoice& pc, int debug)

static ImageField* \\
\textbf{new\_brick\_from\_bounds} (int image\_data\_type, \\
const ImageField& fld\_in, \\
const int* lattice\_bounds, \\
const PrecisionChoice& pc, int debug)

165.3 \\
static ImageField* \textbf{new\_brick} (int image\_data\_type, const ImageField& fld\_in, \\
const ImageBase& grid, const PrecisionChoice& pc, int debug)

\textit{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
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

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
167.1 Basic numbers and numerical vectors.

LimitRange The class template\texttt{class T} class LimitRange (\rightarrow 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

\begin{verbatim}
static T limit_range(char s);
static T limit_range(unsigned int s);
static T limit_range(double s);
\end{verbatim}

The primitive numerical types that can be used for the function parameter or return value are \texttt{char}, \texttt{signed char}, \texttt{unsigned char}, \texttt{short}, \texttt{unsigned short}, \texttt{int}, \texttt{unsigned int}, \texttt{long}, \texttt{unsigned long}, \texttt{float}, \texttt{double} and \texttt{long double}.

TNTVect The class template\texttt{class T} class TNTVect (\rightarrow 40, page 66) inherits from the TNT class \texttt{Vector\texttt{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 \texttt{newsize(int)}.

cast\texttt{to_self_type} The global function template\texttt{class T, class U} T\& cast\texttt{to_self_type(U\& in)} (in namespace Cast\texttt{ToSelfType} (\rightarrow 2, page 12)) is not specifically related to numbers, but it is mentioned here because it is used in the class Number (\rightarrow 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 (\rightarrow 5, page 15).

Number The class Number (\rightarrow 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 Number\texttt{Typed} (\rightarrow 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 instantiation 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 instantiation 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 serves 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-dimensional 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

```cpp
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 `TNTVector<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 coefficients 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 `TNTVector<double> terms`. The function `void make_ntgrl_terms(const double *box)` averages the terms over a rectangle or right parallelepiped, 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 `TNTVector<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

```cpp
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

```cpp
int take_inverse()
```

takes the inverse of the matrix. The function

```cpp
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.

```cpp
void mat_vec_mult(StencilVector const_ref stencil_coeffs_in,
                  StencilVector_ref stencil_coeffs_out) const;
void vec_mat_mult(StencilVector const_ref stencil_coeffs_in,
                  StencilVector_ref stencil_coeffs_out) const;
```
ArbitraryMatrix The class ArbitraryMatrix 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 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

```cpp
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];
invères of the values of _aspect
double inverse_aspect[3];

The default constructor **ImageBase()** can be used and the data members set with the functions

```cpp
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

```cpp
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 global 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
   40 TNTVect

4 LimitRange

18 LinAlgScalar
   5 Number
     11 NumberTyped
     23 LinAlgScalarTyped

30 LinAlgVector
   58 LinAlgVectorSpace
     59 VecSpecificDim
     112 ImageField
       114 ImageFieldTyped

This page was generated with the help of DO C++
http://www.linuxsupport.com/doc++
April 29, 2002 291