## Contents

1. **FDDLib Namespace Index**  
   1.1 FDDLib Namespace List ........................................... 1

2. **FDDLib Hierarchical Index**  
   2.1 FDDLib Class Hierarchy ........................................... 3

3. **FDDLib Compound Index**  
   3.1 FDDLib Compound List ............................................. 7

4. **FDDLib Namespace Documentation**  
   4.1 FDDlib Namespace Reference ................................. 11
   4.2 FDDparse Namespace Reference .............................. 18

5. **FDDLib Class Documentation**  
   5.1 FDDlib::Anomaly2D Class Reference ......................... 21
   5.2 FDDlib::Anomaly3D Class Reference ......................... 24
   5.3 FDDlib::ArbitraryCartesianGrid2D Class Reference ....... 27
   5.4 FDDlib::ArbitraryCartesianGrid2D::s_offsets Struct Reference ................................. 35
   5.5 FDDlib::ArbitraryCartesianGrid3D Class Reference .......... 36
   5.6 FDDlib::ArbitraryCartesianGrid3D::s_offsets Struct Reference ................................. 44
   5.7 FDDlib::ArbitraryDetector2D Class Reference .............. 45
   5.8 FDDlib::ArbitraryDetector2D::s_adentry Struct Reference ................................. 48
   5.9 FDDlib::ArbitraryDetector3D Class Reference .............. 49
   5.10 FDDlib::ArbitraryDetector3D::s_adentry Struct Reference ................................. 52
<table>
<thead>
<tr>
<th>Section</th>
<th>Class/Function Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.11</td>
<td>FDDlib::ArbitrarySource2D Class Reference</td>
</tr>
<tr>
<td>5.12</td>
<td>FDDlib::ArbitrarySource2D::s_asentry Struct Reference</td>
</tr>
<tr>
<td>5.13</td>
<td>FDDlib::ArbitrarySource3D Class Reference</td>
</tr>
<tr>
<td>5.14</td>
<td>FDDlib::ArbitrarySource3D::s_asentry Struct Reference</td>
</tr>
<tr>
<td>5.15</td>
<td>FDDlib::BCDirichlet2D Class Reference</td>
</tr>
<tr>
<td>5.16</td>
<td>FDDlib::BCDirichlet3D Class Reference</td>
</tr>
<tr>
<td>5.17</td>
<td>FDDlib::BCMixed2D Class Reference</td>
</tr>
<tr>
<td>5.18</td>
<td>FDDlib::BCMixed2D::s_neighbordata Struct Reference</td>
</tr>
<tr>
<td>5.19</td>
<td>FDDlib::BCMixed3D Class Reference</td>
</tr>
<tr>
<td>5.20</td>
<td>FDDlib::BCMixed3D::s_neighbordata Struct Reference</td>
</tr>
<tr>
<td>5.21</td>
<td>FDDlib::BCNeumann2D Class Reference</td>
</tr>
<tr>
<td>5.22</td>
<td>FDDlib::BCNeumann3D Class Reference</td>
</tr>
<tr>
<td>5.23</td>
<td>FDDlib::BoundaryCondition2D Class Reference</td>
</tr>
<tr>
<td>5.24</td>
<td>FDDlib::BoundaryCondition3D Class Reference</td>
</tr>
<tr>
<td>5.25</td>
<td>FDDlib::CartesianNode2D Class Reference</td>
</tr>
<tr>
<td>5.26</td>
<td>FDDlib::CartesianNode3D Class Reference</td>
</tr>
<tr>
<td>5.27</td>
<td>FDDlib::CircularAnomaly Class Reference</td>
</tr>
<tr>
<td>5.28</td>
<td>FDDlib::Complex&lt;T&gt; Class Template Reference</td>
</tr>
<tr>
<td>5.29</td>
<td>FDDlib::CubicalAnomaly Class Reference</td>
</tr>
<tr>
<td>5.30</td>
<td>FDDlib::CubicalNode2D Class Reference</td>
</tr>
<tr>
<td>5.31</td>
<td>FDDlib::DenseMatrix&lt;T&gt; Class Template Reference</td>
</tr>
<tr>
<td>5.32</td>
<td>FDDlib::Detector2D Class Reference</td>
</tr>
<tr>
<td>5.33</td>
<td>FDDlib::Detector3D Class Reference</td>
</tr>
<tr>
<td>5.34</td>
<td>FDDlib::DiagonalPreconditioner Class Reference</td>
</tr>
<tr>
<td>5.35</td>
<td>FDDlib::DipoleDetector2D Class Reference</td>
</tr>
<tr>
<td>5.36</td>
<td>FDDlib::DipoleDetector3D Class Reference</td>
</tr>
<tr>
<td>5.37</td>
<td>FDDlib::DipoleSource2D Class Reference</td>
</tr>
<tr>
<td>5.38</td>
<td>FDDlib::DipoleSource3D Class Reference</td>
</tr>
<tr>
<td>5.39</td>
<td>FDDlib::DynamicSparseRowMatrix&lt;T&gt; Class Template Reference</td>
</tr>
<tr>
<td>5.40</td>
<td>FDDlib::EllipsoidalAnomaly Class Reference</td>
</tr>
<tr>
<td>5.41</td>
<td>FDDlib::FiniteDifferences2D Class Reference</td>
</tr>
<tr>
<td>Section</td>
<td>Class/Struct Reference</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------</td>
</tr>
<tr>
<td>5.42</td>
<td>FDDlib::FiniteDifferences2D::s_neighbordata Struct Reference</td>
</tr>
<tr>
<td>5.43</td>
<td>FDDlib::FiniteDifferences3D Class Reference</td>
</tr>
<tr>
<td>5.44</td>
<td>FDDlib::FiniteDifferences3D::s_neighbordata Struct Reference</td>
</tr>
<tr>
<td>5.45</td>
<td>FDDlib::ILU0Preconditioner Class Reference</td>
</tr>
<tr>
<td>5.46</td>
<td>FDDlib::Location2D Class Reference</td>
</tr>
<tr>
<td>5.47</td>
<td>FDDlib::Location3D Class Reference</td>
</tr>
<tr>
<td>5.48</td>
<td>FDDlib::Matrix&lt; T &gt; Class Template Reference</td>
</tr>
<tr>
<td>5.49</td>
<td>FDDlib::MonopoleDetector2D Class Reference</td>
</tr>
<tr>
<td>5.50</td>
<td>FDDlib::MonopoleDetector3D Class Reference</td>
</tr>
<tr>
<td>5.51</td>
<td>FDDlib::MonopoleSource2D Class Reference</td>
</tr>
<tr>
<td>5.52</td>
<td>FDDlib::MonopoleSource3D Class Reference</td>
</tr>
<tr>
<td>5.53</td>
<td>FDDparse::Parser Class Reference</td>
</tr>
<tr>
<td>5.54</td>
<td>FDDparse::Parser::s_arbcartgrid Struct Reference</td>
</tr>
<tr>
<td>5.55</td>
<td>FDDparse::Parser::s_arbitrarydetector Struct Reference</td>
</tr>
<tr>
<td>5.56</td>
<td>FDDparse::Parser::s_arbitrarysource Struct Reference</td>
</tr>
<tr>
<td>5.57</td>
<td>FDDparse::Parser::s_boundary Struct Reference</td>
</tr>
<tr>
<td>5.58</td>
<td>FDDparse::Parser::s_circularanomaly Struct Reference</td>
</tr>
<tr>
<td>5.59</td>
<td>FDDparse::Parser::s_cubicalanomaly Struct Reference</td>
</tr>
<tr>
<td>5.60</td>
<td>FDDparse::Parser::s_dipoledetector Struct Reference</td>
</tr>
<tr>
<td>5.61</td>
<td>FDDparse::Parser::s_dipolesource Struct Reference</td>
</tr>
<tr>
<td>5.62</td>
<td>FDDparse::Parser::s_ellipsoidalanomaly Struct Reference</td>
</tr>
<tr>
<td>5.63</td>
<td>FDDparse::Parser::s_layer Struct Reference</td>
</tr>
<tr>
<td>5.64</td>
<td>FDDparse::Parser::s_monopoledetector Struct Reference</td>
</tr>
<tr>
<td>5.65</td>
<td>FDDparse::Parser::s_monopolesource Struct Reference</td>
</tr>
<tr>
<td>5.66</td>
<td>FDDparse::Parser::s_nebulousgrid Struct Reference</td>
</tr>
<tr>
<td>5.67</td>
<td>FDDparse::Parser::s_node Struct Reference</td>
</tr>
<tr>
<td>5.68</td>
<td>FDDparse::Parser::s_precon Struct Reference</td>
</tr>
<tr>
<td>5.69</td>
<td>FDDparse::Parser::s_property Struct Reference</td>
</tr>
<tr>
<td>5.70</td>
<td>FDDparse::Parser::s_regcartgrid Struct Reference</td>
</tr>
<tr>
<td>5.71</td>
<td>FDDparse::Parser::s_sphericalanomaly Struct Reference</td>
</tr>
<tr>
<td>5.72</td>
<td>FDDparse::Parser::s_squareanomaly Struct Reference</td>
</tr>
<tr>
<td>Section</td>
<td>Class/Reference</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>5.73</td>
<td>FDDlib::Preconditioner Class Reference</td>
</tr>
<tr>
<td>5.74</td>
<td>FDDlib::Property Class Reference</td>
</tr>
<tr>
<td>5.75</td>
<td>FDDlib::RealVector(&lt;T&gt;) Class Template Reference</td>
</tr>
<tr>
<td>5.76</td>
<td>FDDlib::RegularCartesianGrid2D Class Reference</td>
</tr>
<tr>
<td>5.77</td>
<td>FDDlib::RegularCartesianGrid3D Class Reference</td>
</tr>
<tr>
<td>5.78</td>
<td>FDDlib::Source2D Class Reference</td>
</tr>
<tr>
<td>5.79</td>
<td>FDDlib::Source3D Class Reference</td>
</tr>
<tr>
<td>5.80</td>
<td>FDDlib::SparseRowEntry(&lt;T&gt;) Struct Template Reference</td>
</tr>
<tr>
<td>5.81</td>
<td>FDDlib::SparseRowMatrix(&lt;T&gt;) Class Template Reference</td>
</tr>
<tr>
<td>5.82</td>
<td>FDDlib::SphericalAnomaly Class Reference</td>
</tr>
<tr>
<td>5.83</td>
<td>FDDlib::SquareAnomaly Class Reference</td>
</tr>
<tr>
<td>5.84</td>
<td>FDDlib::SSORPreconditioner Class Reference</td>
</tr>
</tbody>
</table>
Chapter 1

FDDLib Namespace Index

1.1 FDDLib Namespace List

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

- FDDlib (Namespace for Frequency Domain Diffusion Library data structures) 11
- FDDparse (Namespace for All Parser Functionality) 18
# Chapter 2

## FDDLib Hierarchical Index

### 2.1 FDDLib Class Hierarchy

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

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDDlib::Anomaly2D</td>
<td>21</td>
</tr>
<tr>
<td>FDDlib::CircularAnomaly</td>
<td>91</td>
</tr>
<tr>
<td>FDDlib::SquareAnomaly</td>
<td>255</td>
</tr>
<tr>
<td>FDDlib::Anomaly3D</td>
<td>24</td>
</tr>
<tr>
<td>FDDlib::CubicalAnomaly</td>
<td>107</td>
</tr>
<tr>
<td>FDDlib::EllipsoidalAnomaly</td>
<td>144</td>
</tr>
<tr>
<td>FDDlib::SphericalAnomaly</td>
<td>252</td>
</tr>
<tr>
<td>FDDlib::ArbitraryCartesianGrid2D</td>
<td>27</td>
</tr>
<tr>
<td>FDDlib::ArbitraryCartesianGrid2D::s_offsets</td>
<td>35</td>
</tr>
<tr>
<td>FDDlib::ArbitraryCartesianGrid3D</td>
<td>36</td>
</tr>
<tr>
<td>FDDlib::ArbitraryCartesianGrid3D::s_offsets</td>
<td>44</td>
</tr>
<tr>
<td>FDDlib::ArbitraryDetector2D::sadentry</td>
<td>48</td>
</tr>
<tr>
<td>FDDlib::ArbitraryDetector3D::sadentry</td>
<td>52</td>
</tr>
<tr>
<td>FDDlib::ArbitrarySource2D::sadentry</td>
<td>56</td>
</tr>
<tr>
<td>FDDlib::ArbitrarySource3D::sadentry</td>
<td>60</td>
</tr>
<tr>
<td>FDDlib::BCMixed2D::sneighbordata</td>
<td>68</td>
</tr>
<tr>
<td>FDDlib::BCMixed3D::sneighbordata</td>
<td>72</td>
</tr>
<tr>
<td>FDDlib::BoundaryCondition2D</td>
<td>75</td>
</tr>
<tr>
<td>FDDlib::BCDirichlet2D</td>
<td>61</td>
</tr>
<tr>
<td>FDDlib::BCMixed2D</td>
<td>65</td>
</tr>
<tr>
<td>FDDlib::BCNeumann2D</td>
<td>73</td>
</tr>
<tr>
<td>FDDlib::BoundaryCondition3D</td>
<td>78</td>
</tr>
<tr>
<td>Class</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>FDDlib::BCDirichlet3D</td>
<td>63</td>
</tr>
<tr>
<td>FDDlib::BCMixed3D</td>
<td>69</td>
</tr>
<tr>
<td>FDDlib::BCNeumann3D</td>
<td>74</td>
</tr>
<tr>
<td>FDDlib::CartesianNode2D</td>
<td>81</td>
</tr>
<tr>
<td>FDDlib::CartesianNode3D</td>
<td>86</td>
</tr>
<tr>
<td>FDDlib::Complex&lt; T &gt;</td>
<td>94</td>
</tr>
<tr>
<td>FDDlib::ComplexVector&lt; T &gt;</td>
<td>101</td>
</tr>
<tr>
<td>FDDlib::Detector2D</td>
<td>114</td>
</tr>
<tr>
<td>FDDlib::ArbitraryDetector2D</td>
<td>45</td>
</tr>
<tr>
<td>FDDlib::DipoleDetector2D</td>
<td>120</td>
</tr>
<tr>
<td>FDDlib::MonopoleDetector2D</td>
<td>181</td>
</tr>
<tr>
<td>FDDlib::Detector3D</td>
<td>116</td>
</tr>
<tr>
<td>FDDlib::ArbitraryDetector3D</td>
<td>49</td>
</tr>
<tr>
<td>FDDlib::DipoleDetector3D</td>
<td>125</td>
</tr>
<tr>
<td>FDDlib::MonopoleDetector3D</td>
<td>184</td>
</tr>
<tr>
<td>FDDlib::FiniteDifferences2D</td>
<td>148</td>
</tr>
<tr>
<td>FDDlib::FiniteDifferences2D::s_neighborhood</td>
<td>158</td>
</tr>
<tr>
<td>FDDlib::FiniteDifferences3D</td>
<td>159</td>
</tr>
<tr>
<td>FDDlib::FiniteDifferences3D::s_neighborhood</td>
<td>169</td>
</tr>
<tr>
<td>FDDlib::Location2D</td>
<td>173</td>
</tr>
<tr>
<td>FDDlib::Location3D</td>
<td>176</td>
</tr>
<tr>
<td>FDDlib::Matrix&lt; T &gt;</td>
<td>179</td>
</tr>
<tr>
<td>FDDlib::DenseMatrix&lt; T &gt;</td>
<td>110</td>
</tr>
<tr>
<td>FDDlib::SparseRowMatrix&lt; T &gt;</td>
<td>248</td>
</tr>
<tr>
<td>FDDlib::DynamicSparseRowMatrix&lt; T &gt;</td>
<td>142</td>
</tr>
<tr>
<td>FDDparse::Parser</td>
<td>195</td>
</tr>
<tr>
<td>FDDparse::Parser::s_arbcartgrid</td>
<td>205</td>
</tr>
<tr>
<td>FDDparse::Parser::s_arbitrarydetector</td>
<td>207</td>
</tr>
<tr>
<td>FDDparse::Parser::s_arbitrarysource</td>
<td>208</td>
</tr>
<tr>
<td>FDDparse::Parser::s_boundary</td>
<td>209</td>
</tr>
<tr>
<td>FDDparse::Parser::s_circularanomaly</td>
<td>210</td>
</tr>
<tr>
<td>FDDparse::Parser::s_cubicalanomaly</td>
<td>211</td>
</tr>
<tr>
<td>FDDparse::Parser::s_dipoledetector</td>
<td>212</td>
</tr>
<tr>
<td>FDDparse::Parser::s_dipolesource</td>
<td>213</td>
</tr>
<tr>
<td>FDDparse::Parser::s_ellipsoidalanomaly</td>
<td>214</td>
</tr>
<tr>
<td>FDDparse::Parser::s_layer</td>
<td>216</td>
</tr>
<tr>
<td>FDDparse::Parser::s_monopoledetector</td>
<td>217</td>
</tr>
<tr>
<td>FDDparse::Parser::s_monopolesource</td>
<td>218</td>
</tr>
<tr>
<td>FDDparse::Parser::s_nebulousgrid</td>
<td>219</td>
</tr>
<tr>
<td>FDDparse::Parser::s_node</td>
<td>220</td>
</tr>
<tr>
<td>FDDparse::Parser::s_precon</td>
<td>221</td>
</tr>
<tr>
<td>FDDparse::Parser::s_property</td>
<td>222</td>
</tr>
<tr>
<td>FDDparse::Parser::s_regcartgrid</td>
<td>223</td>
</tr>
<tr>
<td>FDDparse::Parser::s_sphericalanomaly</td>
<td>224</td>
</tr>
<tr>
<td>FDDparse::Parser::s_squareanomaly</td>
<td>225</td>
</tr>
</tbody>
</table>
2.1 FDDLib Class Hierarchy

FDDlib::Preconditioner .................................................. 226
  FDDlib::DiagonalPreconditioner .................................... 118
  FDDlib::ILU0Preconditioner ....................................... 170
  FDDlib::SSORPreconditioner ...................................... 258
FDDlib::Property ..................................................... 228
FDDlib::RealVector< T > .................................................. 232
  FDDlib::ArbitrarySource2D ....................................... 53
  FDDlib::DipoleSource2D .......................................... 130
  FDDlib::MonopoleSource2D ....................................... 187
FDDlib::Source3D ..................................................... 244
  FDDlib::ArbitrarySource3D ....................................... 57
  FDDlib::DipoleSource3D .......................................... 136
  FDDlib::MonopoleSource3D ....................................... 191
FDDlib::SparseRowEntry< T > ........................................ 247
Chapter 3

FDDLib Compound Index

3.1 FDDLib Compound List

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

- FDDlib::Anomaly2D (Acts as an inhomogeneity in the medium) ........................................ 21
- FDDlib::Anomaly3D (Acts as an inhomogeneity in the medium) ........................................ 24
- FDDlib::ArbitraryCartesianGrid2D (A semi-regular grid) .................................................. 27
- FDDlib::ArbitraryCartesianGrid2D::s::offsets (Struct used for determining how many steps in each direction any node is from the origin) ........................................ 35
- FDDlib::ArbitraryCartesianGrid3D (A semi-regular grid) .................................................. 36
- FDDlib::ArbitraryCartesianGrid3D::s::offsets (Struct used for determining how many steps in each direction any node is from the origin) ........................................ 44
- FDDlib::ArbitraryDetector2D (A detector which includes an arbitrary number of nodes in its aperture) ................................................................. 45
- FDDlib::ArbitraryDetector2D::s::adentry (A single detector entry) .................................... 48
- FDDlib::ArbitraryDetector3D (A detector which includes an arbitrary number of nodes in its aperture) ................................................................. 49
- FDDlib::ArbitraryDetector3D::s::adentry (A single detector entry) .................................... 52
- FDDlib::ArbitrarySource2D (A Source class for which an arbitrary number of locations are covered, each location having an associated complex value) ................................................................. 53
- FDDlib::ArbitrarySource2D::s::asentry (A single source entry) ........................................ 56
- FDDlib::ArbitrarySource3D (A Source class for which an arbitrary number of locations are covered, each location having an associated complex value) ................................................................. 57
- FDDlib::ArbitrarySource3D::s::asentry (A single source entry) ........................................ 60
- FDDlib::BCDirichlet2D (Class to apply the Dirichlet boundary condition \( u=g \)) .......... 61
- FDDlib::BCDirichlet3D (Class to apply the Dirichlet boundary condition \( u=g \)) .......... 63
<table>
<thead>
<tr>
<th>Class/Suite</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDDlib::BCMixed2D</td>
<td>(Class to apply the Mixed boundary condition du/dn+au=g)</td>
<td>65</td>
</tr>
<tr>
<td>FDDlib::BCMixed2D::s_neighbordata</td>
<td>(Struct used for storing information about neighboring nodes)</td>
<td>68</td>
</tr>
<tr>
<td>FDDlib::BCMixed3D</td>
<td>(Class to apply the Mixed boundary condition du/dn+au=g)</td>
<td>69</td>
</tr>
<tr>
<td>FDDlib::BCMixed3D::s_neighbordata</td>
<td>(Struct used for storing information about neighboring nodes)</td>
<td>72</td>
</tr>
<tr>
<td>FDDlib::BCNeumann2D</td>
<td>(Class to apply the Neumann boundary condition du/dn=g)</td>
<td>73</td>
</tr>
<tr>
<td>FDDlib::BCNeumann3D</td>
<td>(Class to apply the Neumann boundary condition du/dn=g)</td>
<td>74</td>
</tr>
<tr>
<td>FDDlib::BoundaryCondition2D</td>
<td>(Abstract base class for boundary conditions)</td>
<td>75</td>
</tr>
<tr>
<td>FDDlib::BoundaryCondition3D</td>
<td>(Abstract base class for boundary conditions)</td>
<td>78</td>
</tr>
<tr>
<td>FDDlib::CartesianNode2D</td>
<td>(Stores characteristics of a node in space)</td>
<td>81</td>
</tr>
<tr>
<td>FDDlib::CartesianNode3D</td>
<td>(Stores characteristics of a node in space)</td>
<td>86</td>
</tr>
<tr>
<td>FDDlib::CircularAnomaly</td>
<td>(Acts as a spherical inhomogeneity in the medium)</td>
<td>91</td>
</tr>
<tr>
<td>FDDlib::Complex&lt;T&gt;</td>
<td>(Templated complex number storage and manipulation class)</td>
<td>94</td>
</tr>
<tr>
<td>FDDlib::ComplexVector&lt;T&gt;</td>
<td>(Class for storing and manipulating vectors of Complex numbers)</td>
<td>101</td>
</tr>
<tr>
<td>FDDlib::CubicalAnomaly</td>
<td>(Acts as a cubical inhomogeneity in the medium)</td>
<td>107</td>
</tr>
<tr>
<td>FDDlib::DenseMatrix&lt;T&gt;</td>
<td>(Routines for storing and manipulating dense matrices)</td>
<td>110</td>
</tr>
<tr>
<td>FDDlib::Detector2D</td>
<td>(The abstract base detector class)</td>
<td>114</td>
</tr>
<tr>
<td>FDDlib::Detector3D</td>
<td>(The abstract base detector class)</td>
<td>116</td>
</tr>
<tr>
<td>FDDlib::DiagonalPreconditioner</td>
<td>(Implement a diagonal preconditioner)</td>
<td>118</td>
</tr>
<tr>
<td>FDDlib::DipoleDetector2D</td>
<td>(A detector collecting data from two weighted nodes)</td>
<td>120</td>
</tr>
<tr>
<td>FDDlib::DipoleDetector3D</td>
<td>(A detector collecting data from two weighted nodes)</td>
<td>125</td>
</tr>
<tr>
<td>FDDlib::DipoleSource2D</td>
<td>(Implement a dipole source)</td>
<td>130</td>
</tr>
<tr>
<td>FDDlib::DipoleSource3D</td>
<td>(Implement a dipole source)</td>
<td>136</td>
</tr>
<tr>
<td>FDDlib::DynamicSparseRowMatrix&lt;T&gt;</td>
<td>(Sparse row-column matrix with dynamic growth)</td>
<td>142</td>
</tr>
<tr>
<td>FDDlib::EllipsoidalAnomaly</td>
<td>(Acts as an ellipsoidal inhomogeneity in the medium)</td>
<td>144</td>
</tr>
<tr>
<td>FDDlib::FiniteDifferences2D</td>
<td>(Implements the finite differences method)</td>
<td>148</td>
</tr>
<tr>
<td>FDDlib::FiniteDifferences2D::s_neighbordata</td>
<td>(Structure used to store info about a node’s neighbors)</td>
<td>158</td>
</tr>
<tr>
<td>FDDlib::FiniteDifferences3D</td>
<td>(Implements the finite differences method)</td>
<td>159</td>
</tr>
<tr>
<td>FDDlib::FiniteDifferences3D::s_neighbordata</td>
<td>(Structure used to store info about a node’s neighbors)</td>
<td>169</td>
</tr>
<tr>
<td>FDDlib::ILU0Preconditioner</td>
<td>(Implement an ILU(0) preconditioner)</td>
<td>170</td>
</tr>
<tr>
<td>FDDlib::Location2D</td>
<td>(A class to store an absolute location in 2-dimensional space)</td>
<td>173</td>
</tr>
</tbody>
</table>
### FDDLib Compound List

- **FDDlib::Location3D** (A class to store an absolute location in 3-dimensional space) ............................................ 176
- **FDDlib::Matrix< T >** (Base class for matrix operations) .................. 179
- **FDDlib::MonopoleDetector2D** (A detector collecting data from a single node) 181
- **FDDlib::MonopoleDetector3D** (A detector collecting data from a single node) 184
- **FDDlib::MonopoleSource2D** (Implement a monopole source) ............ 187
- **FDDlib::MonopoleSource3D** (Implement a monopole source) .......... 191
- **FDDparse::Parser** (A class in which a configuration file can be defined and, once defined, traversed through) ......................... 195
- **FDDparse::Parser::s_arbcartgrid** (Stores all attributes of a semi-regular grid) 205
- **FDDparse::Parser::s_arbitrarydetector** (An Arbitrary Detector Entry) .... 207
- **FDDparse::Parser::s_arbitrarysource** (An Arbitrary Source Entry) ........ 208
- **FDDparse::Parser::s_boundary** (Boundary condition definition (mixedval may be unused)) ........................................... 209
- **FDDparse::Parser::s_circularanomaly** (A Circular Anomaly Entry) ........ 210
- **FDDparse::Parser::s_cubicalanomaly** (A Cubical Anomaly Entry) ......... 211
- **FDDparse::Parser::s_dipoledetector** (A Dipole Detector Entry) ........... 212
- **FDDparse::Parser::s_dipolesource** (A dipole source entry) .............. 213
- **FDDparse::Parser::s_ellipsoidalanomaly** (An Ellipsoidal Anomaly Entry) 214
- **FDDparse::Parser::s_layer** (Data we use to set up a layered background) ... 216
- **FDDparse::Parser::s_monopoledetector** (A Monopole Detector Entry) .... 217
- **FDDparse::Parser::s_monopolesource** (A monopole source entry) ........ 218
- **FDDparse::Parser::s_nebulousgrid** (Stores all attributes of a nebulous grid) 219
- **FDDparse::Parser::s_node** (Data structure to store all attributes of a node) 220
- **FDDparse::Parser::s_precon** (Preconditioner data) ......................... 221
- **FDDparse::Parser::s_property** (A Property Entry) ............................ 222
- **FDDparse::Parser::s_regcartgrid** (Stores all attributes of a regular grid) 223
- **FDDparse::Parser::s_sphericalanomaly** (A Spherical Anomaly Entry) ...... 224
- **FDDparse::Parser::s_squareanomaly** (A Square Anomaly Entry) ........... 225
- **FDDlib::Preconditioner** (Abstract base class for Preconditioners) ........ 226
- **FDDlib::Property** (Medium property class) ...................................... 228
- **FDDlib::RealVector< T >** (Class for storing and manipulating vectors of reals) 232
- **FDDlib::RegularCartesianGrid2D** (An ArbitraryCartesianGrid2D with uniform distance between nodes) ................................. 236
- **FDDlib::RegularCartesianGrid3D** (An ArbitraryCartesianGrid3D with uniform distance between nodes) ................................................................. 238
- **FDDlib::Source2D** (The abstract base source class) ....................... 241
- **FDDlib::Source3D** (The abstract base source class) ....................... 244
- **FDDlib::SparseRowEntry< T >** (A single entry in the SparseRowMatrix) ... 247
- **FDDlib::SparseRowMatrix< T >** (A sparse matrix, with elements stored in row-column form) ............................................... 248
- **FDDlib::SphericalAnomaly** (Acts as a spherical inhomogeneity in the medium) 252
- **FDDlib::SquareAnomaly** (Acts as a cubical inhomogeneity in the medium) 255
- **FDDlib::SSORPreconditioner** (Implement an SSOR preconditioner) ........ 258
Chapter 4

FDDLib Namespace Documentation

4.1 FDDlib Namespace Reference

Namespace for Frequency Domain Diffusion Library data structures.

Compounds

- class Anomaly2D
  
  Acts as an inhomogeneity in the medium.

- class Anomaly3D
  
  Acts as an inhomogeneity in the medium.

- class ArbitraryCartesianGrid2D
  
  A semi-regular grid.

- struct s_offsets
  
  struct used for determining how many steps in each direction any node is from the origin.

- class ArbitraryCartesianGrid3D
  
  A semi-regular grid.
• struct s_offsets
  
  struct used for determining how many steps in each direction any node is from the origin.

• class ArbitraryDetector2D
  
  A detector which includes an arbitrary number of nodes in its aperture.

• struct s_adentry
  
  A single detector entry.

• class ArbitraryDetector3D
  
  A detector which includes an arbitrary number of nodes in its aperture.

• struct s_adentry
  
  A single detector entry.

• class ArbitrarySource2D
  
  A Source class for which an arbitrary number of locations are covered, each location having an associated complex value.

• struct s_asentry
  
  A single source entry.

• class ArbitrarySource3D
  
  A Source class for which an arbitrary number of locations are covered, each location having an associated complex value.

• struct s_asentry
  
  A single source entry.

• class BCDirichlet2D
  
  Class to apply the Dirichlet boundary condition $u=g$.

• class BCDirichlet3D
  
  Class to apply the Dirichlet boundary condition $u=g$.

• class BCMixed2D
  
  Class to apply the Mixed boundary condition $\frac{du}{dn}+au=g$.

• struct s_neighbordata
  
  Struct used for storing information about neighboring nodes.

• class BCMixed3D
4.1 FDDlib Namespace Reference

Class to apply the Mixed boundary condition $\frac{du}{dn}+au=g$.

- struct `s_neighbordata`
  
  _Struct used for storing information about neighboring nodes._

- class `BCNeumann2D`
  
  _Class to apply the Neumann boundary condition $\frac{du}{dn}=g$._

- class `BCNeumann3D`
  
  _Class to apply the Neumann boundary condition $\frac{du}{dn}=g$._

- class `BoundaryCondition2D`
  
  _Abstract base class for boundary conditions._

- class `BoundaryCondition3D`
  
  _Abstract base class for boundary conditions._

- class `CartesianNode2D`
  
  _Stores characteristics of a node in space._

- class `CartesianNode3D`
  
  _Stores characteristics of a node in space._

- class `CircularAnomaly`
  
  _Acts as a spherical inhomogeneity in the medium._

- class `Complex`
  
  _Templated complex number storage and manipulation class._

- class `ComplexVector`
  
  _Class for storing and manipulating vectors of Complex numbers._

- class `CubicalAnomaly`
  
  _Acts as a cubical inhomogeneity in the medium._

- class `DenseMatrix`
  
  _Routines for storing and manipulating dense matrices._

- class `Detector2D`
  
  _The abstract base detector class._

- class `Detector3D`
The abstract base detector class.

- class DiagonalPreconditioner
  
  Implement a diagonal preconditioner.

- class DipoleDetector2D
  
  A detector collecting data from two weighted nodes.

- class DipoleDetector3D
  
  A detector collecting data from two weighted nodes.

- class DipoleSource2D
  
  Implement a dipole source.

- class DipoleSource3D
  
  Implement a dipole source.

- class DynamicSparseRowMatrix
  
  Sparse row-column matrix with dynamic growth.

- class EllipsoidalAnomaly
  
  Acts as an ellipsoidal inhomogeneity in the medium.

- class FiniteDifferences2D
  
  Implements the finite differences method.

- struct s_neighbordata
  
  Structure used to store info about a node's neighbors.

- class FiniteDifferences3D
  
  Implements the finite differences method.

- struct s_neighbordata
  
  Structure used to store info about a node's neighbors.

- class ILU0Preconditioner
  
  Implement an ILU(0) preconditioner.

- class Location2D
  
  A class to store an absolute location in 2-dimensional space.

- class Location3D
4.1 FDDlib Namespace Reference

A class to store an absolute location in 3-dimensional space.

- class **Matrix**
  Base class for matrix operations.

- class **MonopoleDetector2D**
  A detector collecting data from a single node.

- class **MonopoleDetector3D**
  A detector collecting data from a single node.

- class **MonopoleSource2D**
  Implement a monopole source.

- class **MonopoleSource3D**
  Implement a monopole source.

- class **Preconditioner**
  Abstract base class for Preconditioners.

- class **Property**
  Medium property class.

- class **RealVector**
  Class for storing and manipulating vectors of reals.

- class **RegularCartesianGrid2D**
  An **ArbitraryCartesianGrid2D** with uniform distance between nodes.

- class **RegularCartesianGrid3D**
  An **ArbitraryCartesianGrid3D** with uniform distance between nodes.

- class **Source2D**
  The abstract base source class.

- class **Source3D**
  The abstract base source class.

- struct **SparseRowEntry**
  A single entry in the **SparseRowMatrix**.

- class **SparseRowMatrix**
A sparse matrix, with elements stored in row-column form.

- **class SphericalAnomaly**
  Acts as a spherical inhomogeneity in the medium.

- **class SquareAnomaly**
  Acts as a cubical inhomogeneity in the medium.

- **class SSORPreconditioner**
  Implement an SSOR preconditioner.

**Enumerations**

- **enum gridSide2D**
  Enumeration for each side (boundary) of the semi-regular grid.

- **enum gridSide3D**
  Enumeration for each side (boundary) of the semi-regular grid.

- **enum direction2D**
  Directions from a given node.

- **enum direction3D**
  Directions from a given node.

- **enum enum_solver**
  Enumerated type for iterative solver selection.

**Functions**

- **template<class T> double dot (const RealVector<T> &a, const RealVector<T> &b)**
  Compute the dot product of two vectors.

- **template<class T> double norm (const RealVector<T> &rv)**
  Compute the L2 norm of a vector.

- **template<class T> double norm1 (const RealVector<T> &rv)**
  Compute the L1 norm of a vector.
4.1 FDDlib Namespace Reference

- template<class T> RealVector< T > operator * (const T s, const RealVector< T > &rv)
  
  *Multiply a vector by a scalar, allocating space for a new vector.*

**Variables**

- const double EPSILON = 0.0000000000001
  
  *This value is used to account for floating point representation errors.*

**4.1.1 Detailed Description**

Namespace for Frequency Domain Diffusion Library data structures.

**4.1.2 Enumeration Type Documentation**

**4.1.2.1 enum FDDlib::enum_solver**

Enumerated type for iterative solver selection.
- IML\_CG: Conjugate Gradient
- IML\_CGS: Congjugate Gradient Squared
- IML\_BiCGSTAB: BiConjugate Gradient Stabilized
- IML\_GMRES: Generalized Minimum Residual
- IML\_IR: Iterative Refinement

**4.1.3 Variable Documentation**

**4.1.3.1 const double FDDlib::EPSILON = 0.0000000000001**

This value is used to account for floating point representation errors.

Often, when comparing the locations of two points in space, we check that their values are at least as close as the epsilon value, rather than simply checking for equality.
4.2 FDDparse Namespace Reference

Namespace for All Parser Functionality.

Compounds

- class **Parser**
  A class in which a configuration file can be defined and, once defined, traversed through.

- struct **sphericalanomaly**
  A Spherical Anomaly Entry.

- struct **cubicalanomaly**
  A Cubical Anomaly Entry.

- struct **ellipsoidalanomaly**
  An Ellipsoidal Anomaly Entry.

- struct **circularanomaly**
  A Circular Anomaly Entry.

- struct **squareanomaly**
  A Square Anomaly Entry.

- struct **arbitrarilysource**
  An Arbitrary Source Entry.

- struct **monopolesource**
  A monopole source entry.

- struct **dipolesource**
  A dipole source entry.

- struct **arbitrarilydetector**
  An Arbitrary Detector Entry.

- struct **monopoledetector**
  A Monopole Detector Entry.
4.2 FDDparse Namespace Reference

- struct s_dipoledetector
  A Dipole Detector Entry.

- struct s_nebulosgrid
  Stores all attributes of a nebulous grid.

- struct s_regcartgrid
  Stores all attributes of a regular grid.

- struct s_arbcartgrid
  Stores all attributes of a semi-regular grid.

- struct s_node
  Data structure to store all attributes of a node.

- struct s_boundary
  boundary condition definition (mixedval may be unused)

- struct s_property
  A Property Entry.

- struct s_layer
  data we use to set up a layered background

- struct s_precon
  preconditioner data

4.2.1 Detailed Description

Namespace for All Parser Functionality.
Chapter 5

FDDLib Class Documentation

5.1 FDDlib::Anomaly2D Class Reference

Acts as an inhomogeneity in the medium.
#include <Anomaly2D.h>

Inheritance diagram for FDDlib::Anomaly2D:

```
FDDlib::Anomaly2D
 +--------+
 |        |
 |        |
 v        v
FDDlib::CircularAnomaly  FDDlib::SquareAnomaly
```

Public Methods

- `Anomaly2D (Property *property) throw (std::string)`
  
  Constructor.

- virtual `~Anomaly2D ()`

  Destructor.

- void `setProperty (Property *property) throw (std::string)`

  Set the pointer to the Property object associated with this anomaly.
- **Property** * getProperty () const
  
  Get a pointer to the Property object associated with this anomaly.

- virtual bool encloses (const CartesianNode2D &node) const=0
  
  Determine whether this anomaly encloses the given node.

**Protected Attributes**

- **Property** * property_
  
  property associated with this anomaly

**5.1.1 Detailed Description**

Acts as an inhomogeneity in the medium.

This is an abstract base class upon which specific kinds of 2D Anomalies will be implemented.

**Author:**

Kyle Guilbert 7/1/02 @change 12/04/02 Added setProperty

**5.1.2 Constructor & Destructor Documentation**

**5.1.2.1 FDDlib::Anomaly2D::Anomaly2D (Property * property) throw (std::string)**

Constructor.

Parameters:

*property* pointer to the Property object defining the medium type for this Anomaly2D.

**5.1.3 Member Function Documentation**
5.1 FDDlib::Anomaly2D Class Reference

5.1.3.1 virtual bool FDDlib::Anomaly2D::encloses (const CartesianNode2D & node) const [pure virtual]

Determine whether this anomaly encloses the given node.

Parameters:

node the CartesianNode2D

Return values:

true if this Anomaly2D encloses the node
false if this Anomaly2D does noe enclose the node

Implemented in FDDlib::CircularAnomaly, and FDDlib::SquareAnomaly.

5.1.3.2 void FDDlib::Anomaly2D::setProperty (Property * property) throw (std::string)

Set the pointer to the Property object associated with this anomaly.

Exceptions:

string if a NULL pointer is passed

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

- Anomaly2D.h
5.2 FDDlib::Anomaly3D Class Reference

Acts as an inhomogeneity in the medium.

```cpp
#include <Anomaly3D.h>
```

Inheritance diagram for FDDlib::Anomaly3D:

```
FDDlib::Anomaly3D
  `-- FDDlib::CubicalAnomaly
  |    `-- FDDlib::EllipsoidalAnomaly
  |           `-- FDDlib::SphericalAnomaly
```

Public Methods

- `Anomaly3D (Property *property) throw (std::string)`  
  Constructor.

- `virtual ~Anomaly3D ()`
  Destructor.

- `void setProperty (Property *prop) throw (std::string)`
  Set the pointer to the Property object associated with this anomaly.

- `Property * getProperty () const`
  Get a pointer to the Property object associated with this anomaly.

- `virtual bool encloses (const CartesianNode3D &node) const=0`
  Determine whether this anomaly encloses the given node.

Protected Attributes

- `Property * property_
  property associated with this anomaly`
5.2 FDDlib::Anomaly3D Class Reference

5.2.1 Detailed Description

Acts as an inhomogeneity in the medium.
This is an abstract base class upon which specific kinds of 3D Anomalies will be implemented.

Author:
Kyle Guilbert 7/1/02 @change 12/04/02 Added setProperty

5.2.2 Constructor & Destructor Documentation

5.2.2.1 FDDlib::Anomaly3D::Anomaly3D (Property *property) throw (std::string)

Constructor.

Parameters:
prop pointer to the Property object defining the medium type for this Anomaly3D.

5.2.3 Member Function Documentation

5.2.3.1 virtual bool FDDlib::Anomaly3D::encloses (const CartesianNode3D &node) const [pure virtual]

Determine whether this anomaly encloses the given node.

Parameters:
node the node in 3D cartesian space

Return values:
1 if this Anomaly3D encloses the node
0 if this Anomaly3D does not enclose the node

Implemented in FDDlib::CubicalAnomaly, FDDlib::EllipsoidalAnomaly, and FDDlib::SphericalAnomaly.
void FDDlib::Anomaly3D::setProperty (Property * prop) throw (std::string)

Set the pointer to the Property object associated with this anomaly.

Exceptions:

- string if a NULL pointer is passed

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

- Anomaly3D.h
5.3 FDDlib::ArbitraryCartesianGrid2D Class Reference

A semi-regular grid.

#include <ArbitraryCartesianGrid2D.h>

Inheritance diagram for FDDlib::ArbitraryCartesianGrid2D:

- FDDlib::ArbitraryCartesianGrid2D
- FDDlib::RegularCartesianGrid2D

Public Methods

- **ArbitraryCartesianGrid2D** (int numx, int numy, double energySpeed) throw (std::string)
  
  *Constructor.*

- **ArbitraryCartesianGrid2D** (int numx, int numy, double energySpeed, const std::vector<double> &dx, const std::vector<double> &dy) throw (std::string)
  
  *Constructor.*

- virtual ~ArbitraryCartesianGrid2D ()
  
  *Destructor.*

- void **setDelta** (const std::vector<double> &dx, const std::vector<double> &dy) throw (std::string)
  
  *Set the grid distance (delta) lists.*

- std::vector<double> **getDxList** () const
  
  *Returns the DeltaX list.*

- std::vector<double> **getDyList** () const
  
  *Returns the DeltaY list.*

- virtual double **averageDx** () const
Get the average $Dx$ value.

- virtual double \texttt{averageDy} () const
  
  Get the average $Dy$ value.

- int \texttt{numX} () const
  
  number of nodes in the X direction

- int \texttt{numY} () const
  
  number of nodes in the Y direction

- virtual void \texttt{computeNormals} (int maxneighbors, int order, double tol) throw (std::string)
  
  Compute the normal vectors of all boundary nodes.

- virtual void \texttt{analyticalNormals} () throw (std::string)
  
  Compute the normal vectors analytically.

- std::list< CartesianNode2D * > \texttt{getBorderBySide} (gridSide2D side) const
  
  Gets a list of nodes that are on a given side of the grid.

- std::list< CartesianNode2D * > \texttt{getAllBorders} () const
  
  Gets a list of all border nodes.

- int \texttt{offsetsToIndex} (int xoffset, int yoffset) const throw (std::string)
  
  Convert the offset of a node (in 2D) from node zero to its index.

- CartesianNode2D * \texttt{getNode} (int ind) const throw (std::string)
  
  Get a pointer to a node by its index.

- int \texttt{numNodes} () const
  
  Get the number of nodes.

- void \texttt{fillUniformProperty} (Property *prop) throw (std::string)
  
  Fill the grid with a constant property.

- void \texttt{insertAnomaly} (const Anomaly2D &anom)
  
  Insert an anomaly into the grid.

- void \texttt{setEnergySpeed} (double energySpeed)
  
  Set the speed of energy propagation through the grid.

- double \texttt{getEnergySpeed} () const
5.3 FDDlib::ArbitraryCartesianGrid2D Class Reference

Get the speed of energy propagation.

- int numActiveNodes () const throw (std::string)
  Get the number of active nodes in the grid
  **Exceptions:**
  - string if the active node count has not been calculated.

- void generateLinearIndices ()
  Generate linear indices for all nodes.

- int numBorderNodes () const
  Count and return the number of border nodes.

**Static Public Attributes**

- const int DEFAULT_SPACING = 1
  The default unit spacing between nodes (if no delta lists are provided).

**Protected Methods**

- void setupNodes () throw (std::string)
  Destructively sets up the node data to correspond with our delta lists.

**Protected Attributes**

- int numx_
  number of nodes in the x direction

- int numy_
  number of nodes in the y direction

- std::vector< CartesianNode2D > nodeList_
  the list of nodes

- double energySpeed_
  the nominal speed of energy propagation through the medium

- int numActiveNodes_

number of nodes in the grid with an active property

- std::vector<double> deltax
  distances between nodes in x direction
- std::vector<double> deltay
  distances between nodes in y direction

Static Protected Attributes

- const int UNKNOWN_NUMACTIVE = -1
  the number given to numActivePoints when it has not been calculated

5.3.1 Detailed Description

A semi-regular grid.
Nodes are connected in rectangular patterns rather than arbitrarily.

Author:
Kyle Guilbert

5.3.2 Constructor & Destructor Documentation

5.3.2.1 FDDlib::ArbitraryCartesianGrid2D::ArbitraryCartesianGrid2D (int numx, int numy, double energySpeed) throw (std::string)

Constructor.
Since it is unlikely the grid distances will be available at the time of construction, we will set that data later.

Parameters:
  numx number of nodes in the x direction. Range: [1, inf)
  numy number of nodes in the y direction. Range: [1, inf)
  energySpeed nominal speed of the radiation through this grid
5.3 FDDlib::ArbitraryCartesianGrid2D Class Reference

5.3.2.2 FDDlib::ArbitraryCartesianGrid2D::ArbitraryCartesianGrid2D (int numx, int numy, double energySpeed, const std::vector< double > & dx, const std::vector< double > & dy) throw (std::string)

Constructor.

Since it is unlikely the grid distances will be available at the time of construction, we
will set that data later.

Parameters:
- numx number of nodes in the x direction. Range: [1, inf)
- numy number of nodes in the y direction. Range: [1, inf)
- energySpeed nominal speed of the radiation through this grid
- dx deltax list
- dy deltay list

5.3.3 Member Function Documentation

5.3.3.1 virtual void FDDlib::ArbitraryCartesianGrid2D::analyticalNormals () throw (std::string) [virtual]

Compute the normal vectors analytically.

This only works if we are not using irregular boundaries (the condition where the
number of active nodes equals the number of total nodes). This is called from
computeNormals() if this condition is found to be true.

Exceptions:
- string if (numActivePoints() != numPoints()), i.e. if there are any nodes with
  inactive properties in the grid.

5.3.3.2 virtual void FDDlib::ArbitraryCartesianGrid2D::computeNormals (int maxneighbors, int order, double tol) throw (std::string) [virtual]

Compute the normal vectors of all boundary nodes.

Basically, we first populate a sparse matrix giving the Hamming distance of each
boundary node from its neighbors. Then we fit a polynomial surface at that node,
and compute the normal by taking the cross- product of two tangents. Then we verify
that the normal vector is nodeing outwards (reversing direction if it is not).
Author:
   Greg Boverman

Parameters:
   maxneighbors Maximum number of nearest neighbors. Range: [1, inf)
   order order of the polynomial to use. Range: [1, inf)
   tol Tolerance for acceptance of a solution. Range: [0.0, inf) @bugfix 11/12/02
       Normals are now computed correctly.

Exceptions:
   string if any arguments are out of range, or if the estimation was worse than the
       error tolerance for any nodes.

5.3.3.3 void FDDlib::ArbitraryCartesianGrid2D::fillUniformProperty
   (Property * prop) throw (std::string)

Fill the grid with a constant property.

Parameters:
   prop the Property to fill

5.3.3.4 void FDDlib::ArbitraryCartesianGrid2D::generateLinearIndices ()

Generate linear indices for all nodes.

This also calculates numActiveNodes(). @change 12/8/03 Changed name from
  generateLinearIndeces to generateLinearIndices

5.3.3.5 std::list<CartesianNode2D*> FDDlib::ArbitraryCartesianGrid2D::getBorderBySide (gridSide2D side)
   const

Gets a list of nodes that are on a given side of the grid.

Parameters:
   side the grid side we are looking for
5.3.3.6 void FDDlib::ArbitraryCartesianGrid2D::insertAnomaly (const Anomaly2D & anom)

Insert an anomaly into the grid.
This effects the Property of every node in the grid which the anomaly encloses.

Parameters:
- **anom** the Anomaly2D

5.3.3.7 int FDDlib::ArbitraryCartesianGrid2D::offsetsToIndex (int xoffset, int yoffset) const throw (std::string)

Convert the offset of a node (in 2D) from node zero to its index.

Parameters:
- **xoffset** X offset (range: 0 through numX()-1)
- **yoffset** Y offset (range: 0 through numY()-1)

Returns:
position this represents in the node list

Exceptions:
- **string** if invalid parameters or result out of range @bugfix 11/12/02 Parameters are now sufficiently error-checked

5.3.3.8 void FDDlib::ArbitraryCartesianGrid2D::setDelta (const std::vector<double> & dx, const std::vector<double> & dy) throw (std::string)

Set the grid distance (delta) lists.
The number of elements in each array should be one less than the number of nodes in the respective dimension (i.e. dx should have numx-1 elements). Note that these values are copied into this object.

Parameters:
- **dx** DeltaX list. Range for each element: (0.0, inf)
- **dy** DeltaY list. Range for each element: (0.0, inf)

Exceptions:
- **string** if invalid value(s) are found.
The documentation for this class was generated from the following file:

- ArbitraryCartesianGrid2D.h
5.4 FDDlib::ArbitraryCartesianGrid2D::s_offsets Struct Reference

struct used for determining how many steps in each direction any node is from the origin.

#include <ArbitraryCartesianGrid2D.h>

Public Attributes

- int xoffset
  the x offset
- int yoffset
  the y offset

5.4.1 Detailed Description

struct used for determining how many steps in each direction any node is from the origin.

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

- ArbitraryCartesianGrid2D.h
5.5 FDDlib::ArbitraryCartesianGrid3D Class Reference

A semi-regular grid.

#include <ArbitraryCartesianGrid3D.h>

Inheritance diagram for FDDlib::ArbitraryCartesianGrid3D:

```
FDDlib::ArbitraryCartesianGrid3D
   `^^^^^^^^^^^^^^`
FDDlib::RegularCartesianGrid3D
```

Public Methods

- `ArbitraryCartesianGrid3D (int numx, int numy, int numz, double energySpeed)` throw (std::string)
  Constructor.

- `ArbitraryCartesianGrid3D (int numx, int numy, int numz, double energySpeed, const std::vector< double > &dx, const std::vector< double > &dy, const std::vector< double > &dz)` throw (std::string)
  Constructor.

- virtual ~`ArbitraryCartesianGrid3D` ()
  Destructor.

- `void setDelta (const std::vector< double > &dx, const std::vector< double > &dy, const std::vector< double > &dz)` throw (std::string)
  Set the grid distance (delta) lists.

- `std::vector< double > getDxList () const`
  Returns the DeltaX list.

- `std::vector< double > getDyList () const`
  Returns the DeltaY list.

- `std::vector< double > getDzList () const`
Returns the DeltaZ list.

- virtual double averageDx () const
  Get the average Dx value.

- virtual double averageDy () const
  Get the average Dy value.

- virtual double averageDz () const
  Get the average Dz value.

- int numX () const
  number of nodes in the X direction

- int numY () const
  number of nodes in the Y direction

- int numZ () const
  number of nodes in the Z direction

- virtual void computeNormals (int maxneighbors, int order, double tol) throw (std::string)
  Compute the normal vectors of all boundary nodes.

- virtual void analyticalNormals () throw (std::string)
  Compute the normal vectors analytically.

- std::list< CartesianNode3D * > getBorderBySide (gridSide3D side) const
  Gets a list of nodes that are on a given side of the grid.

- std::list< CartesianNode3D * > getAllBorders () const
  Get a list of all border nodes.

- int offsetsToIndex (int xoffset, int yoffset, int zoffset) const throw (std::string)
  Convert the offset of a node (in 3D) from node zero to its index.

- CartesianNode3D * getNode (int ind) const throw (std::string)
  Get a pointer to a node by its index.

- int numNodes () const
  Get the number of nodes.

- void fillUniformProperty (Property *prop) throw (std::string)
Fill the grid with a constant property.

- void insertAnomaly (const Anomaly3D &anom)
  Insert an anomaly into the grid.

- void setEnergySpeed (double energySpeed)
  Set the speed of energy propagation through the grid.

- double getEnergySpeed () const
  Get the speed of energy propagation.

- int numActiveNodes () const throw (std::string)
  Get the number of active nodes in the grid
  Exceptions:
  string if the active node count has not been calculated.

- void generateLinearIndices ()
  Generate linear indices for all nodes. This also calculates numActiveNodes().

- void createLayeredBackground (const std::vector< Property > * &proplist,
  const std::vector< int > &zdepthlist) throw (std::string)
  Create a layered background.

- int numBorderNodes () const
  Count and return the number of border nodes.

Static Public Attributes

- const int DEFAULT_SPACING = 1
  The default unit spacing between nodes (if no delta lists are provided).

Protected Methods

- void setupNodes () throw (std::string)
  Destructively sets up the node data to correspond with our delta lists.
Protected Attributes

- int numx_
  number of nodes in the x direction
- int numy_
  number of nodes in the y direction
- int numz_
  number of nodes in the z direction
- std::vector< CartesianNode3D > nodeList_
  the list of nodes. Each node includes location and property data
- double energySpeed_
  the nominal speed of energy propagation through the medium
- int numActiveNodes_
  number of nodes in the grid with an active property
- std::vector< double > deltax
  distances between nodes in x direction
- std::vector< double > deltay
  distances between nodes in y direction
- std::vector< double > deltaz
  distances between nodes in z direction

Static Protected Attributes

- const int UNKNOWN_NUMACTIVE = -1
  the number given to numActivePoints_ when it has not been calculated

5.5.1 Detailed Description

A semi-regular grid.

Nodes are connected in rectangular patterns rather than arbitrarily.

Author:
  Kyle Guilbert
5.5.2 Constructor & Destructor Documentation

5.5.2.1 FDDlib::ArbitraryCartesianGrid3D::ArbitraryCartesianGrid3D (int numx, int numy, int numz, double energySpeed) throw (std::string)

Constructor.
Since it is unlikely the grid distances will be available at the time of construction, we will set that data later.

Parameters:
numx number of nodes in the x direction. Range: [1, inf)
numy number of nodes in the y direction. Range: [1, inf)
umnz number of nodes in the z direction. Range: [1, inf)
energySpeed nominal speed of the radiation through this grid

5.5.2.2 FDDlib::ArbitraryCartesianGrid3D::ArbitraryCartesianGrid3D (int numx, int numy, int numz, double energySpeed, const std::vector<double>& dx, const std::vector<double>& dy, const std::vector<double>& dz) throw (std::string)

Constructor.
Since it is unlikely the grid distances will be available at the time of construction, we will set that data later.

Parameters:
numx number of nodes in the x direction. Range: [1, inf)
numy number of nodes in the y direction. Range: [1, inf)
umnz number of nodes in the z direction. Range: [1, inf)
energySpeed nominal speed of the radiation through this grid
dx deltax list
dy deltay list
dz deltaz list

5.5.3 Member Function Documentation
5.5.3.1 virtual void FDDlib::ArbitraryCartesianGrid3D::analyticalNormals ()
throw (std::string) [virtual]

Compute the normal vectors analytically.
This only works if we are not using irregular boundaries (the condition where the
number of active nodes equals the number of total nodes). This is called from
computeNormals() if this condition is found to be true.

Exceptions:
    string if (numActivePoints() != numPoints()), i.e. if there are any nodes with
    inactive properties in the grid.

5.5.3.2 virtual void FDDlib::ArbitraryCartesianGrid3D::computeNormals (int maxneighbors, int order, double tol) throw (std::string) [virtual]

Compute the normal vectors of all boundary nodes.
Basically, we first populate a sparse matrix giving the Hamming distance of each
boundary node from its neighbors. Then we fit a polynomial surface at that node,
and compute the normal by taking the cross-product of two tangents. Then we verify
that the normal vector is nodeing outwards (reversing direction if it is not).

Author:
    Greg Boverman

Parameters:
    maxneighbors Maximum number of nearest neighbors. Range: [1, inf)
    order order of the polynomial to use. Range: [1, inf)
    tol Tolerance for acceptance of a solution. Range: [0.0, inf) @bugfix 11/12/02
    Normals are now computed correctly.

Exceptions:
    string if any arguments are out of range, or if the estimation was worse than the
    error tolerance for any nodes.

5.5.3.3 void FDDlib::ArbitraryCartesianGrid3D::createLayeredBackground
(const std::vector< Property* >& proplist, const std::vector<int>& zdepthlist) throw (std::string)

Create a layered background.
These layers are applied consecutively, with each layer having a depth of its corresponding element in zdepthlist.

**Parameters:**
- `proplist` list of property pointers
- `zdepthlist` list of depths (in z direction)

**Exceptions:**
- `string` if vector lengths do not match or if zdepthlist is incomplete (sum of the elements != `numZ()`)

### 5.5.3.4 `void FDDlib::ArbitraryCartesianGrid3D::fillUniformProperty(Property * prop)` throw (std::string)

Fill the grid with a constant property.

**Parameters:**
- `prop` the `Property` to fill

### 5.5.3.5 `std::list<CartesianNode3D*> FDDlib::ArbitraryCartesianGrid3D::getBorderBySide(gridSide3D side)`

Gets a list of nodes that are on a given side of the grid.

**Parameters:**
- `side` the grid side we are looking for

### 5.5.3.6 `void FDDlib::ArbitraryCartesianGrid3D::insertAnomaly(const Anomaly3D & anom)`

Insert an anomaly into the grid.

This effects the `Property` of every node in the grid which the anomaly encloses.

**Parameters:**
- `anom` the `Anomaly3D`
5.5.3.7 int FDDlib::ArbitraryCartesianGrid3D::offsetsToIndex (int xoffset, int yoffset, int zoffset) const throw (std::string)

Convert the offset of a node (in 3D) from node zero to its index.

**Parameters:**
- `xoffset` X offset (range: 0 through `numX()-1`)
- `yoffset` Y offset (range: 0 through `numY()-1`)
- `zoffset` Z offset (range: 0 through `numZ()-1`)

**Returns:**
- position this represents in the node list

**Exceptions:**
- `string` if invalid parameters or result out of range @bugfix 11/12/02 Parameters are now sufficiently error-checked

5.5.3.8 void FDDlib::ArbitraryCartesianGrid3D::setDelta (const std::vector<double>& dx, const std::vector<double>& dy, const std::vector<double>& dz) throw (std::string)

Set the grid distance (delta) lists.

The number of elements in each array should be one less than the number of nodes in the respective dimension (i.e. dx should have numx-1 elements). Note that these values are copied into this object.

**Parameters:**
- `dx` DeltaX list. Range for each element: (0.0, inf)
- `dy` DeltaY list. Range for each element: (0.0, inf)
- `dz` DeltaZ list. Range for each element: (0.0, inf)

**Exceptions:**
- `string` if invalid value(s) are found.

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

- ArbitraryCartesianGrid3D.h
5.6 FDDlib::ArbitraryCartesianGrid3D::s_offsets Struct Reference

struct used for determining how many steps in each direction any node is from the origin.
#include <ArbitraryCartesianGrid3D.h>

Public Attributes

- int xoffset
  the x offset
- int yoffset
  the y offset
- int zoffset
  the z offset

5.6.1 Detailed Description

struct used for determining how many steps in each direction any node is from the origin.

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

- ArbitraryCartesianGrid3D.h
5.7 FDDlib::ArbitraryDetector2D Class Reference

A detector which includes an arbitrary number of nodes in its aperture.

#include <ArbitraryDetector2D.h>

Inheritance diagram for FDDlib::ArbitraryDetector2D:

```
FDDlib::Detector2D
FDDlib::ArbitraryDetector2D
```

Public Methods

- **ArbitraryDetector2D ()**
  Constructor.

- **void addLocation (const Location2D &location, const Complex< double > &weight)**
  Add a node to the aperture.

- **int getApertureSize () const**
  Get the number of nodes in the aperture.

- **s_address getEntry (int entIndex) const throw (std::string)**
  Get entry data given its index into the vector.

- **void setData (const CartesianNode2D &node, const Complex< double > &data)**
  Set the data at this detector.

- **Complex< double > getData () const**
  Get the data stored at this detector.

- **Complex< double > getData (const CartesianNode2D &node) const**
  Get the data stored at this detector, given a node.
Protected Attributes

- std::vector< s_adentry > aperture_
  
  vector of entries in the detector’s aperture

5.7.1 Detailed Description

A detector which includes an arbitrary number of nodes in its aperture.

Author:
Kyle Guilbert

5.7.2 Member Function Documentation

5.7.2.1 void FDDlib::ArbitraryDetector2D::addLocation (const Location2D & location, const Complex<double> & weight)

Add a node to the aperture.

Parameters:

- location  the location of this node in 2D space
- weight  the weight of this node

5.7.2.2 Complex<double> FDDlib::ArbitraryDetector2D::getData (const CartesianNode2D & node) const  [virtual]

Get the data stored at this detector, given a node.

Here we compare the given node’s coordinates with each of the locations in our aperture and return the associated data value.

Parameters:

- node  the node object, in 2D space

Implements FDDlib::Detector2D.
5.7 FDDlib::ArbitraryDetector2D Class Reference 47

5.7.2.3  \texttt{Complex<double>} FDDlib::ArbitraryDetector2D::getData ()  
\hfill [virtual]

Get the data stored at this detector.
Some integration over our included nodes is done here to weight the data.
Implements FDDlib::Detector2D.

5.7.2.4  \texttt{s_adentry} FDDlib::ArbitraryDetector2D::getEntry (int \texttt{entIndex}) const  
\hfill throw (std::string)

Get entry data given its index into the vector.

\textbf{Exceptions:}
\begin{itemize}
  \item \texttt{string} if \texttt{entIndex} is out of range
\end{itemize}

5.7.2.5  void FDDlib::ArbitraryDetector2D::setData (const \texttt{CartesianNode2D} \& \texttt{node}, const \texttt{Complex<double>} \& \texttt{data})  
\hfill [virtual]

Set the data at this detector.
Here we must check each of the nodes in our aperture to determine if they have the 
same location as the given node. If the locations are the same the data is set.

\textbf{Parameters:}
\begin{itemize}
  \item \texttt{node} the node where the data exists, in 2D space
  \item \texttt{data} The complex data @bugfix 11/12/02 Now accounts for floating-node repre-
  \hfill sentation error
\end{itemize}

Implements FDDlib::Detector2D.
The documentation for this class was generated from the following file:

- ArbitraryDetector2D.h
5.8 FDDlib::ArbitraryDetector2D::s_adentry Struct Reference

A single detector entry.

#include <ArbitraryDetector2D.h>

Public Attributes

- Location2D location
  location of this entry, in 2D space

- Complex< double > weight
  weight of this entry

- Complex< double > data
  data stored at this entry

5.8.1 Detailed Description

A single detector entry.

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

- ArbitraryDetector2D.h
5.9 FDDlib::ArbitraryDetector3D Class Reference

A detector which includes an arbitrary number of nodes in its aperture.

#include <ArbitraryDetector3D.h>

Inheritance diagram for FDDlib::ArbitraryDetector3D:

```
FDDlib::Detector3D
    FDDlib::ArbitraryDetector3D
```

Public Methods

- **ArbitraryDetector3D ()**
  _Constructor._

- **void addLocation (const Location3D &location, const Complex<double> &weight)**
  _Add a node to the aperture._

- **int getApertureSize () const**
  _Get the number of nodes in the aperture._

- **s_adentry getEntry (int entIndex) const throw (std::string)**
  _Get entry data given its index into the vector._

- **void setData (const CartesianNode3D &node, const Complex<double> &data)**
  _Set the data at this detector._

- **Complex<double> getData () const**
  _Get the data stored at this detector._

- **Complex<double> getData (const CartesianNode3D &node) const**
  _Get the data stored at this detector, given a node._
Protected Attributes

- std::vector< std::vector< parserentry > > aperture
  vector of entries in the detector's aperture

5.9.1 Detailed Description

A detector which includes an arbitrary number of nodes in its aperture.

Author:
Kyle Guilbert

5.9.2 Member Function Documentation

5.9.2.1 void FDDlib::ArbitraryDetector3D::addLocation (const Location3D & location, const Complex< double > & weight)

Add a node to the aperture.

Parameters:
location the location of this node in 3D space
weight the weight of this node

5.9.2.2 Complex< double > FDDlib::ArbitraryDetector3D::getData (const CartesianNode3D & node) const [virtual]

Get the data stored at this detector, given a node.
Here we compare the given node's coordinates with each of the locations in our aperture and return the associated data value.

Parameters:
node the node object, in 3D space

Implements FDDlib::Detector3D.
5.9.2.3  **Complex<double>** FDDlib::ArbitraryDetector3D::getData ()
[virtual]

Get the data stored at this detector.
Some integration over our included nodes is done here to weight the data.
Implements FDDlib::Detector3D.

5.9.2.4  **s_adentry** FDDlib::ArbitraryDetector3D::getEntry (int entIndex) const
throw (std::string)

Get entry data given its index into the vector.

**Exceptions:**
- **string** if entIndex is out of range

5.9.2.5  **void** FDDlib::ArbitraryDetector3D::setData (const CartesianNode3D &
node, const Complex<double> & data) [virtual]

Set the data at this detector.
Here we must check each of the nodes in our aperture to determine if they have the
same location as the given node. If the locations are the same the data is set.

**Parameters:**
- **node** the node where the data exists, in 3D space
- **data** The complex data @bugfix 11/12/02 Now accounts for floating-node repre-
sentation error

Implements FDDlib::Detector3D.
The documentation for this class was generated from the following file:

- ArbitraryDetector3D.h
5.10 FDDlib::ArbitraryDetector3D::s_adentry Struct Reference

A single detector entry.

#include <ArbitraryDetector3D.h>

Public Attributes

- Location3D location
  location of this entry, in 3D space

- Complex<double> weight
  weight of this entry

- Complex<double> data
  data stored at this entry

5.10.1 Detailed Description

A single detector entry.

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

- ArbitraryDetector3D.h
5.11 FDDlib::ArbitrarySource2D Class Reference

A Source class for which an arbitrary number of locations are covered, each location having an associated complex value.

#include <ArbitrarySource2D.h>

Inheritance diagram for FDDlib::ArbitrarySource2D::

```
FDDlib::Source2D
  |
  V
FDDlib::ArbitrarySource2D
```

Public Methods

- **ArbitrarySource2D ()**
  
  *Default constructor.*

- **ArbitrarySource2D (const std::vector<double> &frequencies)**
  
  *Constructor.*

- **void addLocation (const Location2D &location, const Complex<double> &value)**
  
  *Add a location to the distribution.*

- **int getFieldSize () const**
  
  *Get the number of locations in the source distribution.*

- **s装配式 getEntry (int entIndex) const throw (std::string)**
  
  *Get entry data given its index.*

- **Complex<double> getRightHandSide (const CartesianNode2D &node) const**
  
  *Get the right hand side, given a node of interest.*
Protected Attributes

- std::vector< s,asentry > distribution
  vector of source entries

5.11.1 Detailed Description

A Source class for which an arbitrary number of locations are covered, each location having an associated complex value.

The result is a distribution of right-hand-side values over locations in the Grid.

Author:
  Kyle Guilbert

5.11.2 Constructor & Destructor Documentation

5.11.2.1 FDDlib::ArbitrarySource2D::ArbitrarySource2D (const std::vector< double >& frequencies)

Constructor.

Parameters:
  frequencies vector of frequency modulations. @bugfix 11/7/02 Fixed a bug in the initializer vector that caused a segmentation fault in other functions.

5.11.3 Member Function Documentation

5.11.3.1 void FDDlib::ArbitrarySource2D::addLocation (const Location2D & location, const Complex< double >& value)

Add a location to the distribution.

Parameters:
  location the location of the location, in 2D space
  value Complex right-hand-side value associated with the location
5.11.3.2 \texttt{s\_asentry FDDlib::ArbitrarySource2D::getEntry (int entIndex) const throw (std::string)}

Get entry data given its index.

\textbf{Exceptions:}

\begin{itemize}
\item \texttt{string} if entIndex is out of range
\end{itemize}

5.11.3.3 \texttt{Complex< double > FDDlib::ArbitrarySource2D::getRightHandSide (const CartesianNode2D \& node) const [virtual]} [virtual]

Get the right hand side, given a node of interest.

Here we traverse the vector of entries looking for a location corresponding to the given node’s.

\textbf{Parameters:}

\begin{itemize}
\item \texttt{node} the node (in 2D space) we are concerned with
\end{itemize}

Implements \texttt{FDDlib::Source2D}.

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

- \texttt{ArbitrarySource2D.h}
5.12 FDDlib::ArbitrarySource2D::s_asentry Struct Reference

A single source entry.
#include <ArbitrarySource2D.h>

Public Attributes

- Location2D location
  the location of this entry, in 2D space
- Complex<double> value
  right-hand-side value of this entry

5.12.1 Detailed Description

A single source entry.
The documentation for this struct was generated from the following file:

- ArbitrarySource2D.h
5.13 FDDlib::ArbitrarySource3D Class Reference

A Source class for which an arbitrary number of locations are covered, each location having an associated complex value.

#include <ArbitrarySource3D.h>

Inheritance diagram for FDDlib::ArbitrarySource3D::

```
FDDlib::Source3D
    ↓
FDDlib::ArbitrarySource3D
```

Public Methods

- **ArbitrarySource3D ()**
  
  *Default constructor.*

- **ArbitrarySource3D (const std::vector<double> &frequencies)**
  
  *Constructor.*

- **void addLocation (const Location3D &location, const Complex<double> &value)**
  
  *Add a location to the distribution.*

- **int getFieldSize () const**
  
  *Get the number of locations in the source distribution.*

- **s_asentry getEntry (int entIndex) const throw (std::string)**
  
  *Get entry data given its index.*

- **Complex<double> getRightHandSide (const CartesianNode3D &node)** const
  
  *Get the right hand side, given a node of interest.*
Protected Attributes

- std::vector< s_asentry > distribution_
  vector of source entries

5.13.1 Detailed Description

A Source class for which an arbitrary number of locations are covered, each location having an associated complex value. The result is a distribution of right-hand-side values over locations in the Grid.

Author:
Kyle Guilbert

5.13.2 Constructor & Destructor Documentation

5.13.2.1 FDDlib::ArbitrarySource3D::ArbitrarySource3D (const std::vector< double >& frequencies)

Constructor.

Parameters:
 frequencies vector of frequency modulations. @bugfix 11/7/02 Fixed a bug in the initializer vector that caused a segmentation fault in other functions.

5.13.3 Member Function Documentation

5.13.3.1 void FDDlib::ArbitrarySource3D::addLocation (const Location3D & location, const Complex< double >& value)

Add a location to the distribution.

Parameters:
 location the location of the location, in 3D space
 value Complex right-hand-side value associated with the location
5.13.3.2  

```cpp
s_asentry FDDlib::ArbitrarySource3D::getEntry (int entIndex) const
throw (std::string)
```

Get entry data given its index.

**Exceptions:**

- `string` if entIndex is out of range

5.13.3.3  

```cpp
Complex<double> FDDlib::ArbitrarySource3D::getRightHandSide
(const CartesianNode3D & node) const [virtual]
```

Get the right hand side, given a node of interest.

Here we traverse the vector of entries looking for a location corresponding to the given node’s.

**Parameters:**

- `node` the node (in 3D space) we are concerned with

Implements `FDDlib::Source3D`.

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

- ArbitrarySource3D.h
5.14 FDDlib::ArbitrarySource3D::s_asentry Struct Reference

A single source entry.

#include <ArbitrarySource3D.h>

Public Attributes

- Location3D location
  
  the location of this entry, in 3D space

- Complex<double> value
  
  right-hand-side value of this entry

5.14.1 Detailed Description

A single source entry.

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

- ArbitrarySource3D.h
5.15 FDDlib::BCDirichlet2D Class Reference

Class to apply the Dirichlet boundary condition $u=g$.

```cpp
#include <BCDirichlet2D.h>
```

Inheritance diagram for FDDlib::BCDirichlet2D:

```
FDDlib::BoundaryCondition2D
    `-- FDDlib::BCDirichlet2D
```

**Public Methods**

- virtual void `apply` (const `ArbitraryCartesianGrid2D &grid, SparseRowMatrix< double > &A)` const throw (std::string)
  
  *Apply the Dirichlet boundary condition.*

**5.15.1 Detailed Description**

Class to apply the Dirichlet boundary condition $u=g$.

**Author:**

Kyle Guilbert

**5.15.2 Member Function Documentation**

**5.15.2.1 virtual void FDDlib::BCDirichlet2D::apply (const ArbitraryCartesianGrid2D &grid, SparseRowMatrix< double > & A) const throw (std::string) [virtual]**

Apply the Dirichlet boundary condition.
Parameters:

grid  the 2-dimensional arbitrary cartesian grid we are working with

A    the sparse row matrix

Implements FDDlib::BoundaryCondition2D.

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

- BCDirichlet2D.h
5.16  FDDlib::BCDirichlet3D Class Reference

Class to apply the Dirichlet boundary condition $u=g$.

#include <BCDirichlet3D.h>

Inheritance diagram for FDDlib::BCDirichlet3D:

```
FDDlib::BoundaryCondition3D
  FDDlib::BCDirichlet3D
```

Public Methods

- virtual void apply (const ArbitraryCartesianGrid3D &grid, SparseRowMatrix<double> &A) const throw (std::string)

  Apply the Dirichlet boundary condition.

5.16.1  Detailed Description

Class to apply the Dirichlet boundary condition $u=g$.

Author:
  Kyle Guilbert

5.16.2  Member Function Documentation

5.16.2.1  virtual void FDDlib::BCDirichlet3D::apply (const ArbitraryCartesianGrid3D &grid, SparseRowMatrix<double> &A) const throw (std::string) [virtual]

Apply the Dirichlet boundary condition.
Parameters:
  \textit{grid} the 3-dimensional arbitrary cartesian grid we are working with
  \textit{A} the sparse row matrix

Implements \texttt{FDDlib::BoundaryCondition3D}.

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

\begin{itemize}
  \item BCDirichlet3D.h
\end{itemize}
Class to apply the Mixed boundary condition \( \frac{du}{dn} + au = g \).

#include <BCMixed2D.h>

Inheritance diagram for FDDlib::BCMixed2D:

```
FDDlib::BoundaryCondition2D
 |                  +-- FDDlib::BCMixed2D
 |                  |                  +-- FDDlib::BCNeumann2D
```

### Public Methods

- **BCMixed2D** (double a)
  
  *Constructor.*

- virtual **void apply** (const ArbitraryCartesianGrid2D &grid, SparseRowMatrix<double> &A) const throw (std::string)
  
  *Apply the mixed boundary condition.*

- double **getA** () const
  
  *Get the coefficient \( a \) in the Mixed equation.*

- void **setA** (double a)
  
  *Set the coefficient \( a \) in the Mixed equation.*

### Protected Methods

- bool **activeAndInternal** (const s_neighbordata &ndata) const
  
  *Returns true if the node is active and internal (not a border node).*

- void **resetData** (std::vector<s_neighbordata> &datavec) const
  
  *Resets a given vector of s_neighbordata structures to default values.*
Protected Attributes

- double $a$
  
  *coefficient $a$ in the Mixed equation*

Static Protected Attributes

- const int UNKNOWN_VALUE = -10
  
  *Value used when an index is out of range.*

5.17.1 Detailed Description

Class to apply the Mixed boundary condition $\frac{du}{dn} + au = g$.

**Author:**
Kyle Guilbert @change 12/04/02 Added get/set functions for parameter $a$.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 FDDlib::BCMixed2D::BCMixed2D (double $a$)

Constructor.

**Parameters:**
- $a$ Value of parameter $a$ in Mixed BC equation

5.17.3 Member Function Documentation

5.17.3.1 bool FDDlib::BCMixed2D::activeAndInternal (const s_neighbordata &ndata) const [protected]

Returns true if the node is active and internal (not a border node).
Returns false otherwise.
Parameters:

\texttt{ndata} is \texttt{neighbordata} struct containing neighbor information

5.17.3.2 virtual void FDDlib::BCMixed2D::apply (const
\texttt{ArbitraryCartesianGrid2D} & \texttt{grid}, \texttt{SparseRowMatrix}< \texttt{double} > & \texttt{A})
\texttt{const throw (std::string)} [virtual]

Apply the mixed boundary condition.

Parameters:

\texttt{grid} the 2-dimensional arbitrary cartesian grid we are working with
\texttt{A} the sparse row matrix

Implements \texttt{FDDlib::BoundaryCondition2D}.

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

- \texttt{BCMixed2D.h}
5.18 FDDlib::BCMixed2D::s_neighborhood Struct Reference

Struct used for storing information about neighboring nodes.

#include <BCMixed2D.h>

Public Attributes

- CartesianNode2D * n_node
  
  The neighboring node.

- bool n_borderNode
  
  Whether it is a border node.

- int n_linearIndex
  
  The neighboring node's linear index.

5.18.1 Detailed Description

Struct used for storing information about neighboring nodes.

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

- BCMixed2D.h
5.19 FDDlib::BCMixed3D Class Reference

Class to apply the Mixed boundary condition \( \frac{du}{dn} + au = g \).

```cpp
#include <BCMixed3D.h>
```

Inheritance diagram for FDDlib::BCMixed3D:

```
FDDlib::BoundaryCondition3D
```

```
FDDlib::BCMixed3D
```

```
FDDlib::BCNeumann3D
```

### Public Methods

- **BCMixed3D** (double a)
  
  Constructor.

- **virtual void apply** (const ArbitraryCartesianGrid3D &grid, SparseRowMatrix< double > &A) const throw (std::string)
  
  Apply the mixed boundary condition.

- **double getA** () const
  
  Get the coefficient \( a \) in the Mixed equation.

- **void setA** (double a)
  
  Set the coefficient \( a \) in the Mixed equation.

### Protected Methods

- **bool activeAndInternal** (const s_neighbordata &ndata) const
  
  Returns true if the node is active and internal (not a border node).

- **void resetDataVector** (std::vector< s_neighbordata > &datavec) const
  
  Resets a given vector of s_neighbordata structures to default values.
Protected Attributes

- double $a$
  
  *coefficient $a$ in the Mixed equation*

Static Protected Attributes

- const int UNKNOWN_VALUE = -10
  
  *Value used when an index is out-of-range.*

5.19.1 Detailed Description

Class to apply the Mixed boundary condition $\frac{du}{dn} + au = g$.

**Author:**

Kyle Guilbert @change 12/04/02 Added get/set functions for parameter $a$.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 FDDlib::BCMixed3D::BCMixed3D (double $a$)

Constructor.

**Parameters:**

- $a$ Value of parameter $a$ in Mixed BC equation

5.19.3 Member Function Documentation

5.19.3.1 bool FDDlib::BCMixed3D::activeAndInternal (const s_neighbordata &ndata) const [protected]

Returns true if the node is active and internal (not a border node).
Returns false otherwise.
Parameters:

- **ndata** s.neighbordata struct containing neighbor information

### 5.19.3.2 virtual void FDDlib::BCMixed3D::apply (const ArbitraryCartesianGrid3D & grid, SparseRowMatrix< double > & A) const throw (std::string) [virtual]

Apply the mixed boundary condition.

**Parameters:**

- **grid** the 3-dimensional arbitrary cartesian grid we are working with
- **A** the sparse row matrix

Implements FDDlib::BoundaryCondition3D.

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

- BCMixed3D.h
5.20  **FDDlib::BCMixed3D::s neighbordata Struct Reference**

Struct used for storing information about neighboring nodes.

```cpp
#include <BCMixed3D.h>
```

**Public Attributes**

- `CartesianNode3D * n_node`  
  *The neighboring node.*

- `bool n_borderNode`  
  *Whether it is a border node.*

- `int n_linearIndex`  
  *The neighboring node's linear index.*

5.20.1  **Detailed Description**

Struct used for storing information about neighboring nodes.

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

- `BCMixed3D.h`
5.21 FDDlib::BCNeumann2D Class Reference

Class to apply the Neumann boundary condition $\frac{du}{dn}=g$.

```cpp
#include <BCNeumann2D.h>
```

Inheritance diagram for FDDlib::BCNeumann2D:

```
FDDlib::BoundaryCondition2D
   FDDlib::BCMixed2D
   FDDlib::BCNeumann2D
```

Public Methods

- **BCNeumann2D ()**
  
  *Constructor.*

5.21.1 Detailed Description

Class to apply the Neumann boundary condition $\frac{du}{dn}=g$.

**Author:**

Kyle Guilbert

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

- **BCNeumann2D.h**
5.22  FDDlib::BCNeumann3D Class Reference

Class to apply the Neumann boundary condition $\frac{du}{dn}=g$.

#include <BCNeumann3D.h>

Inheritance diagram for FDDlib::BCNeumann3D:

```
FDDlib::BoundaryCondition3D
  ↓
FDDlib::BCMixed3D
  ↓
FDDlib::BCNeumann3D
```

Public Methods

- **BCNeumann3D ()**
  Constructor.

5.22.1  Detailed Description

Class to apply the Neumann boundary condition $\frac{du}{dn}=g$.

Author:
Kyle Guilbert

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

- BCNeumann3D.h
5.23 FDDlib::BoundaryCondition2D Class Reference

Abstract base class for boundary conditions.
#include <BoundaryCondition2D.h>
Inheritance diagram for FDDlib::BoundaryCondition2D::

```
FDDlib::BoundaryCondition2D
  \|-- FDDlib::BCDirichlet2D
  \|-- FDDlib::BCMixed2D
      \|-- FDDlib::BCNeumann2D
```

Public Methods

- virtual ~BoundaryCondition2D ()

  Destructor.

- virtual void apply (const ArbitraryCartesianGrid2D &grid, SparseRowMatrix<double> &A) const throw (std::string)

  Apply the boundary condition (abstract function).

- virtual void assignToSide (const ArbitraryCartesianGrid2D &grid, gridSide2D side)

  Assign this boundary condition to a given side of the given grid.

- virtual void assignContiguously (const ArbitraryCartesianGrid2D &grid)

  Assign this boundary contiguously to the borders (regular or irregularly defined) of the grid.

- virtual void assignToNode (CartesianNode2D *node) throw (std::string)

  Assign this boundary condition to a single node (effectively adding it to this boundary condition’s scope of interest).
Protected Attributes

- `std::list< CartesianNode2D * > nodes_`  
  *List of nodes to which we will apply this boundary condition.*

5.23.1 Detailed Description

Abstract base class for boundary conditions.

Author:
Kyle Guilbert

5.23.2 Member Function Documentation

5.23.2.1 virtual void FDDlib::BoundaryCondition2D::apply (const 
  ArbitraryCartesianGrid2D & grid, SparseRowMatrix< double > & A) 
  const throw (std::string) [pure virtual]

Apply the boundary condition (abstract function).

Parameters:
- `grid` the arbitrary cartesian grid we are working with
- `A` the sparse row matrix

Implemented in FDDlib::BCDirichlet2D, and FDDlib::BCMixed2D.

5.23.2.2 virtual void FDDlib::BoundaryCondition2D::assignContiguously 
  (const ArbitraryCartesianGrid2D & grid) [virtual]

Assign this boundary contiguously to the borders (regular or irregularly defined) of the grid.

Parameters:
- `grid` the arbitrary cartesian grid we are working with
virtual void FDDlib::BoundaryCondition2D::assignToNode (CartesianNode2D * node) throw (std::string) [virtual]

Assign this boundary condition to a single node (effectively adding it to this boundary condition’s scope of interest).

Parameters:
   node  pointer to the cartesian node

virtual void FDDlib::BoundaryCondition2D::assignToSide (const ArbitraryCartesianGrid2D & grid, gridSide2D side) [virtual]

Assign this boundary condition to a given side of the given grid.

Parameters:
   grid  the arbitrary cartesian grid we are working with
   side  which side of the given grid

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

- BoundaryCondition2D.h
5.24 FDDlib::BoundaryCondition3D Class Reference

Abstract base class for boundary conditions.
#include <BoundaryCondition3D.h>
Inheritance diagram for FDDlib::BoundaryCondition3D::

```
FDDlib::BoundaryCondition3D
  FDDlib::BCDirichlet3D
  FDDlib::BCMixed3D
  FDDlib::BCNeumann3D
```

Public Methods

- virtual ~BoundaryCondition3D ()
  Destructor.

- virtual void apply (const ArbitraryCartesianGrid3D &grid, SparseRowMatrix<double> &A) const throw (std::string)
  Apply the boundary condition (abstract function).

- virtual void assignToSide (const ArbitraryCartesianGrid3D &grid, gridSide3D side)
  Assign this boundary condition to a given side of the given grid.

- virtual void assignContiguously (const ArbitraryCartesianGrid3D &grid)
  Assign this boundary contiguously to the borders (regular or irregularly defined) of the grid.

- virtual void assignToNode (CartesianNode3D *node) throw (std::string)
  Assign this boundary condition to a single node (effectively adding it to this boundary condition’s scope of interest).
Protected Attributes

- std::list< CartesianNode3D * > nodes

List of nodes to which we will apply this boundary condition.

5.24.1 Detailed Description

Abstract base class for boundary conditions.

Author:
Kyle Guilbert

5.24.2 Member Function Documentation

5.24.2.1 virtual void FDDlib::BoundaryCondition3D::apply (const ArbitraryCartesianGrid3D & grid, SparseRowMatrix< double > & A) const throw (std::string) [pure virtual]

Apply the boundary condition (abstract function).

Parameters:
- grid the arbitrary cartesian grid we are working with
- A the sparse row matrix

Implemented in FDDlib::BCDirichlet3D, and FDDlib::BCMixed3D.

5.24.2.2 virtual void FDDlib::BoundaryCondition3D::assignContiguously (const ArbitraryCartesianGrid3D & grid) [virtual]

Assign this boundary contiguously to the borders (regular or irregularly defined) of the grid.

Parameters:
- grid the arbitrary cartesian grid we are working with
5.24.2.3 virtual void FDDlib::BoundaryCondition3D::assignToNode
(CartesianNode3D * node) throw (std::string) [virtual]

Assign this boundary condition to a single node (effectively adding it to this boundary condition’s scope of interest).

Parameters:
node pointer to the cartesian node

5.24.2.4 virtual void FDDlib::BoundaryCondition3D::assignToSide (const ArbitraryCartesianGrid3D & grid, gridSide3D side) [virtual]

Assign this boundary condition to a given side of the given grid.

Parameters:
grid the arbitrary cartesian grid we are working with
side which side of the given grid

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

- BoundaryCondition3D.h
5.25 FDDlib::CartesianNode2D Class Reference

Stores characteristics of a node in space.

#include <CartesianNode2D.h>

Public Methods

- **CartesianNode2D** (const Location2D &location) throw (std::string)
  Constructor.

- **Location2D** getLocation () const
  Get the location of this node.

- **int** getNumNeighbors () const
  Get the number of neighbors surrounding this node.

- **void** setNeighbor (direction2D dir, CartesianNode2D *node) throw (std::string)
  Set the neighbor of this node.

- **CartesianNode2D** * getNeighbor (direction2D dir) const throw (std::string)
  Get a neighbor of this node.

- **void** setProperty (Property *prop)
  Set the medium property associated with this node.

- **Property** * getProperty () const
  Get a pointer to the Property object associated with this node.

- **void** setNormalVector (const Location2D &normalVector)
  Set the normal vector of this node.

- **Location2D** getNormalVector () const
  Get the normal vector of this node.

- **double** distanceTo (const CartesianNode2D &node) const
  Get the distance to another node.

- **bool** compareLocation (const CartesianNode2D &node) const
Determine whether this \texttt{CartesianNode2D} has the same location as the given \texttt{CartesianNode2D}.

- \textbf{bool} \texttt{isBorderNode} () \textbf{const}
  
  Returns whether this node is on the border, based on neighbor data.

- \textbf{void} \texttt{setLinearIndex} (int linearIndex) \textbf{throw} (std::string)
  
  Sets the index this node corresponds to in the system of equations.

- \textbf{int} \texttt{getLinearIndex} () \textbf{const} \textbf{throw} (std::string)
  
  Get the linear index.

\textbf{Static Public Attributes}

- \textbf{const} \textbf{int} \texttt{INACTIVE\_LIN\_INDEX} = -1
  
  the value given to \texttt{linearIndex} when it is not in the system of equations.

\textbf{Protected Methods}

- \textbf{bool} \texttt{activeNeighbor} (direction2D \texttt{dir}) \textbf{const}
  
  Returns whether the neighbor node in the given direction has an active property.

\textbf{Protected Attributes}

- \textbf{Location2D} \texttt{location}\_
  
  location of this node in 2D space

- std::vector< CartesianNode2D * > \texttt{neighbors}\_
  
  neighbor pointers

- \textbf{Property} * \texttt{property}\_
  
  \textit{Property} associated with this node.

- \textbf{Location2D} \texttt{normalVector}\_
  
  the location that forms our normal vector

- \textbf{int} \texttt{linearIndex}\_
  
  index of this node in the system of equations
Static Protected Attributes

- const int UNKNOWN_INDEX = -2
  
  *the value given to linearIndex when it has not been defined*

5.25.1 Detailed Description

Stores characteristics of a node in space.

Author:
Kyle Guilbert

5.25.2 Constructor & Destructor Documentation

5.25.2.1 FDDlib::CartesianNode2D::CartesianNode2D (const Location2D & location) throw (std::string)

Constructor.

Parameters:
  
  *location* the location of this node

5.25.3 Member Function Documentation

5.25.3.1 bool FDDlib::CartesianNode2D::activeNeighbor (direction2D dir)

Returns whether the neighbor node in the given direction has an active property.

Parameters:
  
  *dir* the direction of interest
5.25.3.2 bool FDDlib::CartesianNode2D::compareLocation (const CartesianNode2D & node) const

Determine whether this CartesianNode2D has the same location as the given CartesianNode2D.

Parameters:
   node the node we are comparing to

Return values:
   true if the locations are the same
   false if the locations are not the same

5.25.3.3 double FDDlib::CartesianNode2D::distanceTo (const CartesianNode2D & node) const

Get the distance to another node.

Parameters:
   node the node to which we would like to find the distance.

5.25.3.4 int FDDlib::CartesianNode2D::getLinearIndex () throw (std::string)

Get the linear index.

Exceptions:
   string if the linear index is undefined.

5.25.3.5 CartesianNode2D* FDDlib::CartesianNode2D::getNeighbor (direction2D dir) const throw (std::string)

Get a neighbor of this node.

Parameters:
   dir direction of interest

Exceptions:
   string if neighbor is nonexistent
5.25.3.6 void FDDlib::CartesianNode2D::setLinearIndex (int linearIndex) throw (std::string)

Sets the index this node corresponds to in the system of equations.

Parameters:
  linearIndex the linear index. Range: [0, inf), as well as INACTIVE_INDEX

Exceptions:
  string if linearIndex is out of range

5.25.3.7 void FDDlib::CartesianNode2D::setNeighbor (direction2D dir,
  CartesianNode2D * node) throw (std::string)

Set the neighbor of this node.

Parameters:
  dir the direction the neighbor is located at
  node pointer to the neighboring node

5.25.3.8 void FDDlib::CartesianNode2D::setNormalVector (const Location2D & normalVector)

Set the normal vector of this node.

Parameters:
  normalVector the location representing the other end of the outward normal vector from this node.

5.25.3.9 void FDDlib::CartesianNode2D::setProperty (Property * prop)

Set the medium property associated with this node.

Parameters:
  prop pointer to the Property object

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

- CartesianNode2D.h
5.26  FDDlib::CartesianNode3D Class Reference

Stores characteristics of a node in space.

#include <CartesianNode3D.h>

Public Methods

- **CartesianNode3D** (const Location3D &location) throw (std::string)
  
  Constructor.

- **Location3D getLocation () const**
  
  Get the location of this node.

- **int getNumNeighbors () const**
  
  Get the number of neighbors surrounding this node.

- **void setNeighbor (direction3D dir, CartesianNode3D *node) throw (std::string)**
  
  Set the neighbor of this node.

- **CartesianNode3D * getNeighbor (direction3D dir) const throw (std::string)**
  
  Get a neighbor of this node.

- **void setProperty (Property *prop)**
  
  Set the medium property associated with this node.

- **Property * getProperty () const**
  
  Get a pointer to the Property object associated with this node.

- **void setNormalVector (const Location3D &normalVector)**
  
  Set the normal vector of this node.

- **Location3D getNormalVector () const**
  
  Get the normal vector of this node.

- **double distanceTo (const CartesianNode3D &node) const**
  
  Get the distance to another node.

- **bool compareLocation (const CartesianNode3D &node) const**
Determine whether this `CartesianNode3D` has the same location as the given `CartesianNode3D`.

- `bool isBorderNode() const`
  
  Returns whether this node is bordering, based on neighbor data.

- `void setLinearIndex(int linearIndex) throw (std::string)`
  
  Sets the index this node corresponds to in the system of equations.

- `int getLinearIndex() const throw (std::string)`
  
  Get the linear index.

### Static Public Attributes

- `const int INACTIVE_LINEAR_INDEX = -1`
  
  the value given to linearIndex when it is not in the system of equations.

### Protected Methods

- `bool activeNeighbor(direction3D dir) const`
  
  Returns whether the neighbor node in the given direction has an active property.

### Protected Attributes

- `Location3D location_`
  
  location of this node in 3D space

- `std::vector< CartesianNode3D * > neighbors_`
  
  neighbor pointers

- `Property * property_`
  
  Property associated with this node.

- `Location3D normalVector_`
  
  the location that forms our normal vector

- `int linearIndex_`
  
  index of this node in the system of equations
Static Protected Attributes

- `const int UNKNOWN_LIN_INDEX = -2`  
  *the value given to linearIndex when it has not been defined*

5.26.1 Detailed Description

Stores characteristics of a node in space.

Author:
Kyle Guilbert

5.26.2 Constructor & Destructor Documentation

5.26.2.1 `FDDlib::CartesianNode3D::CartesianNode3D (const Location3D & location)` throw (std::string)

Constructor.

Parameters:
- `location` the location of this node

5.26.3 Member Function Documentation

5.26.3.1 `bool FDDlib::CartesianNode3D::activeNeighbor (direction3D dir)` const [protected]

Returns whether the neighbor node in the given direction has an active property.

Parameters:
- `dir` the direction of interest
5.26.3.2 bool FDDlib::CartesianNode3D::compareLocation (const CartesianNode3D & node) const

Determine whether this CartesianNode3D has the same location as the given CartesianNode3D.

Parameters:
  node the node we are comparing to

Return values:
  true if the locations are the same
  false if the locations are not the same

5.26.3.3 double FDDlib::CartesianNode3D::distanceTo (const CartesianNode3D & node) const

Get the distance to another node.

Parameters:
  node the node to which we would like to find the distance.

5.26.3.4 int FDDlib::CartesianNode3D::getLinearIndex () throw (std::string)

Get the linear index.

Exceptions:
  string if the linear index is undefined.

5.26.3.5 CartesianNode3D* FDDlib::CartesianNode3D::getNeighbor (direction3D dir) const throw (std::string)

Get a neighbor of this node.

Parameters:
  dir direction of interest

Exceptions:
  string if neighbor is nonexistent

Generated on Mon Aug 30 15:41:16 2004 for FDDLib by Doxygen
5.26.3.6 void FDDlib::CartesianNode3D::setLinearIndex (int linearIndex) throw (std::string)

Sets the index this node corresponds to in the system of equations.

**Parameters:**

- **linearIndex** - the linear index. Range: [0, inf), as well as INACTIVE\_IN\_INDEX

**Exceptions:**

- **string** - if linearIndex is out of range

5.26.3.7 void FDDlib::CartesianNode3D::setNeighbor (direction3D dir, CartesianNode3D * node) throw (std::string)

Set the neighbor of this node.

**Parameters:**

- **dir** - the direction the neighbor is located at
- **node** - pointer to the neighboring node

5.26.3.8 void FDDlib::CartesianNode3D::setNormalVector (const Location3D & normalVector)

Set the normal vector of this node.

**Parameters:**

- **normalVector** - the location representing the other end of the outward normal vector from this node.

5.26.3.9 void FDDlib::CartesianNode3D::setProperty (Property * prop)

Set the medium property associated with this node.

**Parameters:**

- **prop** - pointer to the Property object

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

- CartesianNode3D.h
5.27 FDDlib::CircularAnomaly Class Reference

Acts as a spherical inhomogeneity in the medium.

#include <CircularAnomaly.h>

Inheritance diagram for FDDlib::CircularAnomaly::

```
FDDlib::Anomaly2D

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FDDlib::CircularAnomaly</td>
</tr>
</tbody>
</table>
```

Public Methods

- **CircularAnomaly** (const Location2D &center, double radius, Property *prop)
  throw (std::string)
  
  Constructor.

- **Location2D getCenter** () const
  
  Get the location of this spherical anomaly.

- void **setCenter** (const Location2D &center)
  
  Set the location of this spherical anomaly.

- double **getRadius** () const
  
  Get the radius of this spherical anomaly.

- void **setRadius** (double radius) throw (std::string)
  
  Set the radius of this spherical anomaly.

- bool **encloses** (const CartesianNode2D &node) const
  
  Determine whether this circle encloses the given node.

Protected Attributes

- **Location2D center_**
Detailed Description

Acts as a spherical inhomogeneity in the medium.

Author:
Kyle Guilbert 7/30/02, Derek Uluski @change 12/04/02 Added a bunch of set/get functions.

Constructor & Destructor Documentation

FDDlib::CircularAnomaly::CircularAnomaly (const Location2D & center, double radius, Property * prop) throw (std::string)

Constructor.

Parameters:
- center location in 2D space of the circle’s center
- radius radius of circle (range: positive values)
- prop pointer to Property object associated with the circle

Member Function Documentation

FDDlib::CircularAnomaly::encloses (const CartesianNode2D & node) const [virtual]

Determine whether this circle encloses the given node.

Parameters:
- node the node of interest

Return values:
- true if this circle encloses the CartesianNode2D
false if this circle does not enclose the CartesianNode2D

Implements FDDlib::Anomaly2D.

5.27.3.2 void FDDlib::CircularAnomaly::setCenter (const Location2D & center)

Set the location of this spherical anomaly.

Parameters:
  center location in 2D space of the circle’s center

5.27.3.3 void FDDlib::CircularAnomaly::setRadius (double radius) throw (std::string)

Set the radius of this spherical anomaly.

Parameters:
  radius the radius. Range: [0.0, inf)

Exceptions:
  string if radius is out of range

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

- CircularAnomaly.h
5.28 FDDlib::Complex< T > Class Template Reference

Templated complex number storage and manipulation class.
#include <Complex.h>

Public Methods

- **Complex ()**
  
  Constructor.

- **~Complex ()**
  
  Destructor.

- **Complex (const T realPart, const T imagPart)**
  
  Constructor.

- **Complex (const T t)**
  
  Constructor for a real number.

- **Complex (const Complex< T >&cn)**
  
  Copy constructor.

- **Complex< T > & operator= (const Complex< T >&c)**
  
  Assignment operator overloading.

- **T getReal () const**
  
  Get the real part.

- **T getImag () const**
  
  Get the imaginary part.

- **void setReal (T r)**
  
  Set the real part.

- **void setImag (T i)**
  
  Set the imaginary part.
- **Complex\(<T>\ &\ operator+= (const Complex\(<T>\ &c)\)**  
  *Addition assignment operator.*

- **Complex\(<T>\ &\ operator-= (const Complex\(<T>\ &c)\)**  
  *Subtraction assignment operator.*

- **Complex\(<T>\ &\ operator*= (const Complex\(<T>\ &c)\)**  
  *Multiplication assignment operator.*

- **Complex\(<T>\ &\ operator/= (const Complex\(<T>\ &c)\)**  
  *Division assignment operator.*

- **Complex\(<T>\ &\ operator/= (const T s)\)**  
  *Division assignment operator.*

- **Complex\(<T>\ &\ operator+ (const Complex\(<T>\ &c)\)** const  
  *Addition operator.*

- **Complex\(<T>\ &\ operator- (const Complex\(<T>\ &c)\)** const  
  *Subtraction operator.*

- **Complex\(<T>\ &\ operator* (const Complex\(<T>\ &c)\)** const  
  *Multiplication operator.*

- **Complex\(<T>\ &\ operator* (const T s)\)** const  
  *Multiplication operator.*

- **Complex\(<T>\ &\ operator/ (const Complex\(<T>\ &c)\)** const  
  *Division operator.*

- **Complex\(<T>\ &\ operator/ (const T s)\)** const  
  *Division operator.*

- **double magnitude () const\**  
  *Compute the magnitude.*

- **double squareMagnitude () const\**  
  *Compute the square magnitude.*
• Complex< T >  conjugate () const

   Get the conjugate of this Complex number.

Protected Attributes

• T realPart_

   Real part of the number.

• T imagPart_

   Imaginary part of the number.

5.28.1 Detailed Description

template<class T> class FDDlib::Complex< T >

Tempated complex number storage and manipulation class.

Author:

   Greg Boverman , Kyle Guilbert @change 1/15/04: Added a scalar division opera-
   tor. Jennifer Black

5.28.2 Constructor & Destructor Documentation

5.28.2.1 template<class T> FDDlib::Complex< T >::Complex (const T
realPart, const T imagPart)

Constructor.

Parameters:

   realPart  the real part

   imagPart  the imaginary part
5.28 FDDlib::Complex< T > Class Template Reference

5.28.2.2 template <class T> FDDlib::Complex< T >::Complex (const T t)

Constructor for a real number.

Parameters:
   t  real part

Note:
   imaginary part will be 0

5.28.3 Member Function Documentation

5.28.3.1 template <class T> Complex< T > FDDlib::Complex< T >::operator *
   (const T s) const

Multiplication operator.

Parameters:
   s  A scalar value

5.28.3.2 template <class T> Complex< T > FDDlib::Complex< T >::operator *
   (const Complex< T >& c) const

Multiplication operator.

Parameters:
   c  A complex number

5.28.3.3 template <class T> Complex< T > & FDDlib::Complex< T >
   ::operator *= (const T s)

Multiplication assignment operator.

Parameters:
   s  A scalar value
5.28.3.4 template<class T> Complex< T > & FDDlib::Complex< T >::operator *= (const Complex< T > & c)

Multiplication assignment operator.

Parameters:
  - c  A complex number

5.28.3.5 template<class T> Complex< T > FDDlib::Complex< T >::operator+ (const Complex< T > & c) const

Addition operator.

Parameters:
  - c  A complex number

5.28.3.6 template<class T> Complex< T > & FDDlib::Complex< T >::operator+= (const Complex< T > & c)

Addition assignment operator.

Parameters:
  - c  A complex number

5.28.3.7 template<class T> Complex< T > FDDlib::Complex< T >::operator- (const Complex< T > & c) const

Subtraction operator.

Parameters:
  - c  A complex number
5.28.3.8  template<class T> Complex<T> & FDDlib::Complex<T> ::operator-= (const Complex<T> & c)

Subtraction assignment operator.

**Parameters:**
- c A complex number

5.28.3.9  template<class T> Complex<T> FDDlib::Complex<T> ::operator/ (const T s) const

Division operator.

**Parameters:**
- s A scalar value

**Note:**
1/15/04 Added by J. Black

5.28.3.10  template<class T> Complex<T> & FDDlib::Complex<T> ::operator/ (const Complex<T> & c) const

Division operator.

**Parameters:**
- c A complex number

5.28.3.11  template<class T> Complex<T> & FDDlib::Complex<T> ::operator/= (const T s)

Division assignment operator.

**Parameters:**
- s A scalar value
5.28.3.12  template<class T> Complex< T > & FDDlib::Complex< T >::operator/= (const Complex< T > & c)

Division assignment operator.

Parameters:
  - c  A complex number

5.28.3.13  template<class T> Complex< T > & FDDlib::Complex< T >::operator=(const Complex< T > & c)

Assignment operator overloading.

Parameters:
  - c  A Complex number

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

- Complex.h
Class for storing and manipulating vectors of Complex numbers.

#include <ComplexVector.h>

Public Methods

- **ComplexVector** (int len)
  Constructor.

- **ComplexVector** ()
  Default Constructor.

- void **init** (int len)
  *Initialize this vector to the given length.*

- **ComplexVector** (const ComplexVector& cv)
  Copy Constructor.

- ~ComplexVector ()
  Destructor.

- void **deallocate** ()
  *Dealocate this vector’s memory.*

- Complex< T > **val** (int i) const
  *Get a Complex value from this vector.*

- Complex< T >& **val** (int i)
  *Get a Complex value from this vector.*

- Complex< T > operator() (int i) const
  *operator() overloader*

- Complex< T >& operator() (int i)
  *operator() overloader*
• void set (int ind, Complex< T >& c)
  Set a value in the vector.

• void setReal (int ind, T real)
  Set a real value.

• void setImag (int ind, T imag)
  Set an imaginary value.

• T getReal (int ind) const
  Get the real component at the given index.

• T getImag (int ind) const
  Get the imaginary component at the given index.

• int size () const
  Get the size of this vector.

• ComplexVector< T >& operator= (const ComplexVector< T >& cv)
  Assignment overloading.

• ComplexVector< T >& operator += (const Complex< T >& c)
  Multiplication assignment overloading.

• ComplexVector< T >& operator += (const ComplexVector< T >& cv)
  Addition assignment overloading.

• ComplexVector< T >& operator -= (const ComplexVector< T >& cv)
  Subtraction assignment overloading.

• ComplexVector< T > operator+ (const ComplexVector< T >& cv) const
  Addition overloading.

• ComplexVector< T > operator- (const ComplexVector< T >& cv) const
  Subtraction overloading.

• ComplexVector< T > operator* (const Complex< T >& c) const
  Multiplication overloading.

• ComplexVector< T > conjugate () const
  Get the Complex conjugate of this vector.
5.29 FDDlib::ComplexVector< T > Class Template Reference

- Complex< T > dot (const ComplexVector< T > cv) const
  Calculate the dot product between this and another ComplexVector.

- double norm () const
  Compute the norm of this vector.

Protected Attributes

- int len_
  Length of the vector.

- Complex< T > * valVec_
  List of values.

5.29.1 Detailed Description

template<class T> class FDDlib::ComplexVector< T >

Class for storing and manipulating vectors of Complex numbers.

Author:
Greg Boverman 11-27-01, Kyle Guilbert

5.29.2 Constructor & Destructor Documentation

5.29.2.1 template<class T> FDDlib::ComplexVector< T >::ComplexVector
  (int len)

Constructor.

Parameters:
  len  Length of the vector to allocate.

5.29.3 Member Function Documentation
5.29.3.1 template<class T> ComplexVector<T> FDDlib::ComplexVector<T>::operator* (const Complex<T> & c) const

Multiplication overloading.

Parameters:
  
  c  A Complex number

5.29.3.2 template<class T> ComplexVector<T> & FDDlib::ComplexVector<T>::operator*=(const Complex<T> & c)

Multiplication assignment overloading.

Parameters:
  
  c  A Complex number

5.29.3.3 template<class T> ComplexVector<T> FDDlib::ComplexVector<T>::operator+ (const ComplexVector<T> & cv) const

Addition overloading.

Parameters:
  
  cv  A ComplexVector

5.29.3.4 template<class T> ComplexVector<T> & FDDlib::ComplexVector<T>::operator+=(const ComplexVector<T> & cv)

Addition assignment overloading.

Parameters:
  
  cv  A ComplexVector
5.29.3.5 template<class T> ComplexVector<T> FDDlib::ComplexVector<T> ::operator- (const ComplexVector<T> & cv) const

Subtraction overloading.

Parameters:
  cv A ComplexVector

5.29.3.6 template<class T> ComplexVector<T> & FDDlib::ComplexVector<T> ::operator= (const ComplexVector<T> & cv)

Subtraction assignment overloading.

Parameters:
  cv A ComplexVector

5.29.3.7 template<class T> ComplexVector<T> & FDDlib::ComplexVector<T> ::operator= (const ComplexVector<T> & cv)

Assignment overloading.

Parameters:
  cv A ComplexVector

5.29.3.8 template<class T> void FDDlib::ComplexVector<T> ::set (int ind, Complex<T> & c) [inline]

Set a value in the vector.

Parameters:
  ind Index
  c A Complex number
5.29.3.9  template<class T> void FDDlib::ComplexVector<T>::setImag (int ind, T imag) [inline]

Set an imaginary value.

Parameters:
   ind  Index
   imag Scalar imaginary value

5.29.3.10 template<class T> void FDDlib::ComplexVector<T>::setReal (int ind, T real) [inline]

Set a real value.

Parameters:
   ind  Index
   real Scalar real value

5.29.3.11 template<class T> Complex<T> & FDDlib::ComplexVector<T>::val (int i) [inline]

Get a Complex value from this vector.

Returns:
   Non-constant method returns a reference to the array element

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

- ComplexVector.h
5.30 FDDlib::CubicalAnomaly Class Reference

Acts as a cubical inhomogeneity in the medium.
#include <CubicalAnomaly.h>
Inheritance diagram for FDDlib::CubicalAnomaly:

```
FDDlib::Anomaly3D
|   |
V   V
|   |
FDDlib::CubicalAnomaly
```

Public Methods

- **CubicalAnomaly** (const Location3D &center, double length, Property *prop) throw (std::string)
  
  *Constructor.*

- void **setCenter** (const Location3D &center)

  *Set the location of this cubical anomaly.*

- **Location3D getCenter** () const

  *Get the location of this cubical anomaly.*

- double **getLength** () const

  *Get the length of this cubical anomaly.*

- void **setLength** (double length) throw (std::string)

  *Set the length of this cubical anomaly.*

- bool **encloses** (const CartesianNode3D &node) const

  *Determine whether this cube encloses the given node.*

Protected Attributes

- **Location3D center_**
5.30.1 Detailed Description

Acts as a cubical inhomogeneity in the medium.

Author: Kyle Guilbert 7/30/02, Derek Uluski @change 12/04/02 Added lots of set/get functions

5.30.2 Constructor & Destructor Documentation

5.30.2.1 FDDlib::CubicalAnomaly::CubicalAnomaly (const Location3D & center, double length, Property * prop) throw (std::string)

Constructor.

Parameters:
- `center` location in 3D space of the cube’s center
- `length` length of each side. Range: [0.0, inf)
- `prop` pointer to the Property object

5.30.3 Member Function Documentation

5.30.3.1 bool FDDlib::CubicalAnomaly::encloses (const CartesianNode3D & node) const [virtual]

Determine whether this cube encloses the given node.

Parameters:
- `node` the node of interest

Return values:
- `true` if this cube encloses the CartesianNode3D
0 if this cube does not enclose the CartesianNode3D

Implements FDDlib::Anomaly3D.

5.30.3.2 void FDDlib::CubicalAnomaly::setCenter (const Location3D & center)

Set the location of this cubical anomaly.

Parameters:
   location in 3D space of the cube’s center

5.30.3.3 void FDDlib::CubicalAnomaly::setLength (double length) throw (std::string)

Set the length of this cubical anomaly.

Parameters:
   length the length of the sides of the cube. Range: [0.0, inf)

Exceptions:
   string if length is out of range

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

- CubicalAnomaly.h
5.31 FDDlib::DenseMatrix< T > Class Template Reference

Routines for storing and manipulating dense matrices.

```cpp
#include <DenseMatrix.h>
```

Inheritance diagram for FDDlib::DenseMatrix< T >:

```
FDDlib::Matrix< T >
```

```
FDDlib::DenseMatrix< T >
```

### Public Methods

- `DenseMatrix (int rows, int cols)`
  Constructor.

- `DenseMatrix ()`
  Constructor.

- `DenseMatrix (const DenseMatrix< T > &dm)`
  Copy constructor.

- `virtual ~DenseMatrix ()`
  Destructor.

- `void deallocate ()`
  Deallocate this matrix.

- `void init (int rows, int cols)`
  Initialize this matrix.

- `DenseMatrix< T > & operator= (const DenseMatrix< T > &dm)`
  Set this dense matrix to another dense matrix.

- `DenseMatrix< T > & operator *= (const T v)`
Multiply this matrix by another dense matrix and set the result to this matrix.

- template<class Vector> Vector operator* (const Vector &m) const
  Multiply this matrix by a vector and return the result.

- int getNumRows() const
  Get the number of rows.

- int getNumCols() const
  Get the number of columns.

- T val (int row, int col) const
  Get the value at a row and column.

- void set(int row, int col, T v) throw (std::string)
  Set the value at a row and column.

- void QRDecomp(DenseMatrix<T> &Q, DenseMatrix<T> &R) const throw (std::string)
  Do a QR decomposition of an overdetermined matrix by modified Gram-Schmidt orthogonalization.

- template<class Vector> void LSSolve(Vector &x, const Vector &b) const throw (std::string)
  Do a least-squares solve of the overdetermined system Ax = b.

- DenseMatrix<T> operator* (const DenseMatrix<T> &a) const
  Multiply a matrix, creating a new one.

- DenseMatrix<T> operator+ (const DenseMatrix<T> &a) const
  Subtract a matrix, creating a new one.

- DenseMatrix<T> operator- (const DenseMatrix<T> &a) const
  Subtract a matrix, creating a new one.

- DenseMatrix<T> transpose() const
  Get the transposed version of this matrix.

- RealVector<T> getRow(int row) const
  Get an entire row of the matrix.

- RealVector<T> getColumn(int col) const
  Get an entire column of the matrix.
T & operator() (int row, int col)

*element access using () operator*

T operator() (int row, int col) const

*element access using () operator (const version)*

**Protected Attributes**

- int rows_
  
  *number of rows*

- int cols_
  
  *number of columns*

- T * valVec_
  
  *matrix values*

### 5.31.1 Detailed Description

template<class T> class FDDlib::DenseMatrix<T>

Routines for storing and manipulating dense matrices.

**Author:**

Greg Boverman 1-28-2002

### 5.31.2 Constructor & Destructor Documentation

#### 5.31.2.1 template<class T> FDDlib::DenseMatrix<T>::DenseMatrix (int rows, int cols)

Constructor.

**Parameters:**

- *rows* number of rows
- *cols* number of cols
5.31.3 Member Function Documentation

5.31.3.1 template<class T> void FDDlib::DenseMatrix<T>::init (int rows, int cols)

Initialize this matrix.

Parameters:
  - rows number of rows.
  - cols number of cols.

5.31.3.2 template<class T> template<class Vector> void FDDlib::DenseMatrix<T>::LSSolve (Vector &x, const Vector &b) const throw (std::string)

Do a least-squares solve of the overdetermined system Ax = b.

Parameters:
  - x Vector "x" in system Ax = b
  - b Vector "b" in system Ax = b

5.31.3.3 template<class T> void FDDlib::DenseMatrix<T>::QRDecomp (DenseMatrix<T> &Q, DenseMatrix<T> &R) const throw (std::string)

Do a QR decomposition of an overdetermined matrix by modified Gram-Schmidt orthogonalization.

Parameters:
  - Q resulting orthogonal matrix
  - R resulting upper triangular matrix

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

- DenseMatrix.h
5.32  FDDlib::Detector2D Class Reference

The abstract base detector class.

```cpp
#include <Detector2D.h>
```

Inheritance diagram for FDDlib::Detector2D:

```
FDDlib::Detector2D
```

```
FDDlib::ArbitraryDetector2D  FDDlib::DipoleDetector2D  FDDlib::MonopoleDetector2D
```

**Public Methods**

- virtual `~Detector2D()`
  
  *Destructor.*

- virtual void `setData(const CartesianNode2D &node, const Complex<double> &data)=0`
  
  *Set the data at this detector.*

- virtual `Complex<double> getData() const=0`
  
  *Get the data stored at this detector.*

- virtual `Complex<double> getData(const CartesianNode2D &node) const=0`
  
  *Get the data stored at this detector, given a node.*

5.32.1  Detailed Description

The abstract base detector class.

Here we provide an interface by which data can be stored and retrieved from a specific type of receiver.

**Author:**

Kyle Guilbert
5.32.2 Member Function Documentation

5.32.2.1 virtual Complex<double> FDDlib::Detector2D::getData (const CartesianNode2D & node) const [pure virtual]

Get the data stored at this detector, given a node.
This method includes functionality to determine whether this detector is concerned with this node, and returning the associated data value.

Parameters:
- node the node of interest (in 2D space)

Implemented in FDDlib::ArbitraryDetector2D, FDDlib::DipoleDetector2D, and FDDlib::MonopoleDetector2D.

5.32.2.2 virtual void FDDlib::Detector2D::setData (const CartesianNode2D & node, const Complex<double> & data) [pure virtual]

Set the data at this detector.
Here we must check whether this detector is actually concerned with the given node (i.e. does the detector’s aperture include the node).

Parameters:
- node the node, in 2D space, where the data exists
- data The complex data

Implemented in FDDlib::ArbitraryDetector2D, FDDlib::DipoleDetector2D, and FDDlib::MonopoleDetector2D.

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

- Detector2D.h
5.33 FDDlib::Detector3D Class Reference

The abstract base detector class.

#include <Detector3D.h>

Inheritance diagram for FDDlib::Detector3D::

```
FDDlib::Detector3D
FDDlib::ArbitraryDetector3D   FDDlib::DipoleDetector3D   FDDlib::MonopoleDetector3D
```

Public Methods

- virtual ~Detector3D ()
  Destructor.

- virtual void setData (const CartesianNode3D &node, const Complex<double> &data)=0
  Set the data at this detector.

- virtual Complex<double> getData () const=0
  Get the data stored at this detector.

- virtual Complex<double> getData (const CartesianNode3D &node) const=0
  Get the data stored at this detector, given a node.

5.33.1 Detailed Description

The abstract base detector class.

Here we provide an interface by which data can be stored and retrieved from a specific type of receiver.

Author:
Kyle Guilbert
5.33.2 Member Function Documentation

5.33.2.1 virtual Complex<double> FDDlib::Detector3D::getData (const CartesianNode3D & node) const [pure virtual]

Get the data stored at this detector, given a node.
This method includes functionality to determine whether this detector is concerned with this node, and returning the associated data value.

Parameters:
- node the node of interest (in 3D space)

Implemented in FDDlib::ArbitraryDetector3D, FDDlib::DipoleDetector3D, and FDDlib::MonopoleDetector3D.

5.33.2.2 virtual void FDDlib::Detector3D::setData (const CartesianNode3D & node, const Complex<double> & data) [pure virtual]

Set the data at this detector.
Here we must check whether this detector is actually concerned with the given node (i.e. does the detector’s aperture include the node).

Parameters:
- node the node, in 3D space, where the data exists
- data The complex data

Implemented in FDDlib::ArbitraryDetector3D, FDDlib::DipoleDetector3D, and FDDlib::MonopoleDetector3D.

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

- Detector3D.h
5.34 FDDlib::DiagonalPreconditioner Class Reference

Implement a diagonal preconditioner.

#include <DiagonalPreconditioner.h>

Inheritance diagram for FDDlib::DiagonalPreconditioner:

```
FDDlib::Preconditioner
  FDDlib::DiagonalPreconditioner
```

Public Methods

- **DiagonalPreconditioner (SparseRowMatrix<double> &SRMD)**
  
  *constructor*

- **DiagonalPreconditioner ()**
  
  *constructor*

- **void init (SparseRowMatrix<double> &SRMD) throw (std::string)**
  
  *initialize preconditioner data*

- **~DiagonalPreconditioner ()**
  
  *destructor*

- **RealVector<double> solve (const RealVector<double> &x) const**
  
  *solve the system*

5.34.1 Detailed Description

Implement a diagonal preconditioner.

**Author:**

Greg Boverman 8-29-01 @bugfix 11/17/02 (Kyle Guilbert) Fixed some old ExtensibleSparseRowMatrix declarations.
5.34.2 Constructor & Destructor Documentation

5.34.2.1 FDDlib::DiagonalPreconditioner::DiagonalPreconditioner
(SparseRowMatrix< double > & SRMD) [inline]

class constructor

Parameters:
   SRMD our Sparse Row matrix

5.34.3 Member Function Documentation

5.34.3.1 void FDDlib::DiagonalPreconditioner::init (SparseRowMatrix<
  double > & SRMD) throw (std::string) [inline, virtual]

initialize preconditioner data

Parameters:
   SRMD out Sparse Row Matrix

Implements FDDlib::Preconditioner.

5.34.3.2 RealVector< double > FDDlib::DiagonalPreconditioner::solve (const
  RealVector< double > & x) const [inline, virtual]

solve the system

Parameters:
   x right hand side

Return values:
   the solution vector

Implements FDDlib::Preconditioner.

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

- DiagonalPreconditioner.h
5.35 FDDlib::DipoleDetector2D Class Reference

A detector collecting data from two weighted nodes.

#include <DipoleDetector2D.h>

Inheritance diagram for FDDlib::DipoleDetector2D:

```
FDDlib::Detector2D
    ↓
FDDlib::DipoleDetector2D
```

Public Methods

- **DipoleDetector2D** (const Location2D &location1, const Complex< double > &weight1, const Location2D &location2, const Complex< double > &weight2)
  
  Constructor.

- void **setLocations** (const Location2D &location1, const Location2D &location2)

  Set the locations of (or nodes covered by) this detector.

- void **setWeights** (const Complex< double > &weight1, const Complex< double > &weight2)

  Set the weights of the nodes covered by this detector.

- void **getLocations** (Location2D &location1, Location2D &location2) const

  Get the locations of (or nodes covered by) this detector.

- void **getWeights** (Complex< double > &weight1, Complex< double > &weight2) const

  Get the weights of the nodes covered by this detector.

- void **setData** (const CartesianNode2D &node, const Complex< double > &data)

  Set the data at this detector.
• Complex< double > getData() const  
  *Get the data stored at this detector. Here we do a simple weighted sum.*

• Complex< double > getData(const CartesianNode2D &node) const  
  *Get the data stored at this detector, given a node.*

### Protected Attributes

- Location2D location1_  
  *location of node 1*

- Complex< double > weight1_  
  *weight of node 1*

- Complex< double > data1_  
  *data stored at node 1*

- Location2D location2_  
  *location of node 2*

- Complex< double > weight2_  
  *weight of node 2*

- Complex< double > data2_  
  *data stored at node 2*

### 5.35.1 Detailed Description

A detector collecting data from two weighted nodes.

**Author:**
  Kyle Guilbert

### 5.35.2 Constructor & Destructor Documentation
5.35.2.1 FDDlib::DipoleDetector2D::DipoleDetector2D (const Location2D & location1, const Complex< double > & weight1, const Location2D & location2, const Complex< double > & weight2)

Constructor.

Parameters:
- location1 location of node 1
- weight1 weight of node 1
- location2 location of node 2
- weight2 weight of node 2 @bugfix 11/26/02 Fixed a bug in the constructor list in the implementation.

5.35.3 Member Function Documentation

5.35.3.1 Complex< double > FDDlib::DipoleDetector2D::getData (const CartesianNode2D & node) const [virtual]

Get the data stored at this detector, given a node.
Here we compare the node’s coordinates with the coordinates of both our poles.

Parameters:
- node the node of interest (in 2D space)

Implements FDDlib::Detector2D.

5.35.3.2 void FDDlib::DipoleDetector2D::getLocations (Location2D & location1, Location2D & location2) const

Get the locations of (or nodes covered by) this detector.
This is done via reference parameters.

Parameters:
- location1 location of node 1
- location2 location of node 2
5.35.3.3 void FDDlib::DipoleDetector2D::getWeights (Complex< double > & weight1, Complex< double > & weight2) const

Get the weights of the nodes covered by this detector.
This is done via reference parameters.

Parameters:
weight1 weight of node 1
weight2 weight of node 2

5.35.3.4 void FDDlib::DipoleDetector2D::setData (const CartesianNode2D & node, const Complex< double > & data) [virtual]

Set the data at this detector.
Here we check whether the given node has the same location as either of the locations in our aperture.

Parameters:
node the node, in 2D space, where the data exists
data The complex data @bugfix 11/12/02 Now accounts for floating-node representation error

Implements FDDlib::Detector2D.

5.35.3.5 void FDDlib::DipoleDetector2D::setLocations (const Location2D & location1, const Location2D & location2)

Set the locations of (or nodes covered by) this detector.

Parameters:
location1 location of node 1
location2 location of node 2

5.35.3.6 void FDDlib::DipoleDetector2D::setWeights (const Complex< double > & weight1, const Complex< double > & weight2)

Set the weights of the nodes covered by this detector.
Parameters:

- \textit{weight1}: weight of node 1
- \textit{weight2}: weight of node 2

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

- DipoleDetector2D.h
5.36  FDDlib::DipoleDetector3D Class Reference

A detector collecting data from two weighted nodes.

#include <DipoleDetector3D.h>

Inheritance diagram for FDDlib::DipoleDetector3D:

```
FDDlib::Detector3D
  |    
  v    
FDDlib::DipoleDetector3D
```

### Public Methods

- **DipoleDetector3D** (const Location3D &location1, const Complex< double > &weight1, const Location3D &location2, const Complex< double > &weight2)
  
  Constructor.

- **void setLocations** (const Location3D &location1, const Location3D &location2)
  
  Set the locations of (or nodes covered by) this detector.

- **void setWeights** (const Complex< double > &weight1, const Complex< double > &weight2)
  
  Set the weights of the nodes covered by this detector.

- **void getLocations** (Location3D &location1, Location3D &location2) const
  
  Get the locations of (or nodes covered by) this detector.

- **void getWeights** (Complex< double > &weight1, Complex< double > &weight2) const
  
  Get the weights of the nodes covered by this detector.

- **void setData** (const CartesianNode3D &node, const Complex< double > &data)
  
  Set the data at this detector.
- **Complex< double > getData () const**
  
  *Get the data stored at this detector. Here we do a simple weighted sum.*

- **Complex< double > getData (const CartesianNode3D &node) const**
  
  *Get the data stored at this detector, given a node.*

### Protected Attributes

- **Location3D location1_**
  
  *location of node 1*

- **Complex< double > weight1_**
  
  *weight of node 1*

- **Complex< double > data1_**
  
  *data stored at node 1*

- **Location3D location2_**
  
  *location of node 2*

- **Complex< double > weight2_**
  
  *weight of node 2*

- **Complex< double > data2_**
  
  *data stored at node 2*

### 5.36.1 Detailed Description

A detector collecting data from two weighted nodes.

**Author:**

Kyle Guilbert

### 5.36.2 Constructor & Destructor Documentation
5.36.1 FDDlib::DipoleDetector3D::DipoleDetector3D (const Location3D & location1, const Complex< double > & weight1, const Location3D & location2, const Complex< double > & weight2)

Constructor.

Parameters:
- location1 location of node 1
- weight1 weight of node 1
- location2 location of node 2
- weight2 weight of node 2 @bugfix 11/26/02 Fixed a bug in the constructor list in the implementation.

5.36.3 Member Function Documentation

5.36.3.1 Complex< double > FDDlib::DipoleDetector3D::getData (const CartesianNode3D & node) const [virtual]

Get the data stored at this detector, given a node.

Here we compare the node’s coordinates with the coordinates of both our poles.

Parameters:
- node the node of interest (in 3D space)

Implements FDDlib::Detector3D.

5.36.3.2 void FDDlib::DipoleDetector3D::getLocations (Location3D & location1, Location3D & location2) const

Get the locations of (or nodes covered by) this detector.

This is done via reference parameters.

Parameters:
- location1 location of node 1
- location2 location of node 2
5.36.3.3 void FDDlib::DipoleDetector3D::getWeights (Complex< double > & weight1, Complex< double > & weight2) const

Get the weights of the nodes covered by this detector.
This is done via reference parameters.

Parameters:
weight1 weight of node 1
weight2 weight of node 2

5.36.3.4 void FDDlib::DipoleDetector3D::setData (const CartesianNode3D & node, const Complex< double > & data) [virtual]

Set the data at this detector.
Here we check whether the given node has the same location as either of the locations in our aperture.

Parameters:
node the node, in 3D space, where the data exists
data The complex data @bugfix 11/12/02 Now accounts for floating-node representation error

Implements FDDlib::Detector3D.

5.36.3.5 void FDDlib::DipoleDetector3D::setLocations (const Location3D & location1, const Location3D & location2)

Set the locations of (or nodes covered by) this detector.

Parameters:
location1 location of node 1
location2 location of node 2

5.36.3.6 void FDDlib::DipoleDetector3D::setWeights (const Complex< double > & weight1, const Complex< double > & weight2)

Set the weights of the nodes covered by this detector.
Parameters:

- `weight1` weight of node 1
- `weight2` weight of node 2

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

- DipoleDetector3D.h
5.37 FDDlib::DipoleSource2D Class Reference

Implement a dipole source.

```
#include <DipoleSource2D.h>
```

Inheritance diagram for FDDlib::DipoleSource2D:

```
FDDlib::DipoleSource2D
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FDDlib::Source2D</td>
</tr>
</tbody>
</table>
```

Public Methods

- **DipoleSource2D** (const Location2D &location1, double int1, double phase1, const Location2D &location2, double int2, double phase2)
  
  Constructor.

- **DipoleSource2D** (const Location2D &location1, double int1, double phase1, const Location2D &location2, double int2, double phase2, const std::vector<double> &frequencies)
  
  Constructor.

- void **set** (const Location2D &location1, double int1, double phase1, const Location2D &location2, double int2, double phase2)
  
  Set the attributes of this DipoleSource2D.

- void **getPoles** (Location2D &location1, double &int1, double &phase1, Location2D &location2, double &int2, double &phase2) const
  
  Get the poles of this DipoleSource2D by getting all the data directly.

- void **setLocations** (const Location2D &location1, const Location2D &location2)
  
  Set the locations of this dipole source.

- void **getLocations** (Location2D &location1, Location2D &location2) const
  
  Get the locations of both poles via reference parameters.
5.37 FDDlib::DipoleSource2D Class Reference

- void **getIntensities** (double &int1, double &int2) const
  
  Get the intensities of both poles via reference parameters.

- void **setIntensities** (double int1, double int2)
  
  Set the intensities of both poles.

- void **getPhases** (double &phase1, double &phase2) const
  
  Get the phase angles of both poles via reference parameters.

- void **setPhases** (double phase1, double phase2)
  
  Set the phase angles of both poles.

- **Complex<double>** getRightHandSide (const **CartesianNode2D** &node)
  
  Get the right-hand-side of the PDE equation focusing on the given node in the grid.

**Protected Attributes**

- **Location2D** location1
  
  location of the first pole

- double **int1**
  
  Intensity of first pole.

- double **phase1**
  
  Phase of first pole.

- **Location2D** location2
  
  location of the second pole

- double **int2**
  
  Intensity of second pole.

- double **phase2**
  
  Phase of second pole.

5.37.1 Detailed Description

Implement a dipole source.

This is essentially two node sources.
Author:
Kyle Guilbert @change 11/27/02 Removed any interaction with MonopoleSource2D. This functionality was rather useless. @change 12/04/02 Added set functions for phase and intensity.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 FDDlib::DipoleSource2D::DipoleSource2D (const Location2D & location1, double int1, double phase1, const Location2D & location2, double int2, double phase2)

Constructor.

Parameters:
location1 location of the first pole
int1 intensity of the first pole
phase1 phase of the first pole
location2 location of the second pole
int2 intensity of the second pole
phase2 phase of the second pole

5.37.2.2 FDDlib::DipoleSource2D::DipoleSource2D (const Location2D & location1, double int1, double phase1, const Location2D & location2, double int2, double phase2, const std::vector< double > & frequencies)

Constructor.

Parameters:
location1 location of the first pole
int1 intensity of the first pole
phase1 phase of the first pole
location2 location of the second pole
int2 intensity of the second pole
phase2 phase of the second pole
frequencies list of modulation frequencies
numFreq number of frequencies in the list (range: positive integers)
5.37.3 Member Function Documentation

5.37.3.1 void FDDlib::DipoleSource2D::getIntensities (double & int1, double & int2) const

Get the intensities of both poles via reference parameters.

Parameters:

- int1 intensity of the first pole
- int2 intensity of the second pole

5.37.3.2 void FDDlib::DipoleSource2D::getLocations (Location2D & location1, Location2D & location2) const

Get the locations of both poles via reference parameters.

Parameters:

- location1 location of the first pole
- location2 location of the second pole

5.37.3.3 void FDDlib::DipoleSource2D::getPhases (double & phase1, double & phase2) const

Get the phase angles of both poles via reference parameters.

Parameters:

- phase1 phase of the first pole
- phase2 phase of the second pole

5.37.3.4 void FDDlib::DipoleSource2D::getPoles (Location2D & location1, double & int1, double & phase1, Location2D & location2, double & int2, double & phase2) const

Get the poles of this DipoleSource2D by getting all the data directly.
Parameters:

- **location1**  location of the first pole
- **int1**  intensity of the first pole
- **phase1**  phase of the first pole
- **location2**  location of the second pole
- **int2**  intensity of the second pole
- **phase2**  phase of the second pole

### 5.37.3.5 Complex< double > FDDlib::DipoleSource2D::getRightHandSide
(const CartesianNode2D & node) const  [virtual]

Get the right-hand-side of the PDE equation focusing on the given node in the grid.

Parameters:

- **node**  the node of interest (in 2D space)

Returns:

complex right-hand-side

Implements FDDlib::Source2D.

### 5.37.3.6 void FDDlib::DipoleSource2D::set (const Location2D & location1, double int1, double phase1, const Location2D & location2, double int2, double phase2)

Set the attributes of this DipoleSource2D.

Parameters:

- **location1**  location of the first pole
- **int1**  intensity of the first pole
- **phase1**  phase of the first pole
- **location2**  location of the second pole
- **int2**  intensity of the second pole
- **phase2**  phase of the second pole
5.37.3.7  void FDDlib::DipoleSource2D::setIntensities (double int1, double int2)

Set the intensities of both poles.

Parameters:
  int1  intensity of the first pole
  int2  intensity of the second pole

5.37.3.8  void FDDlib::DipoleSource2D::setLocations (const Location2D & location1, const Location2D & location2)

Set the locations of this dipole source.

Parameters:
  location1  location of the first pole
  location2  location of the second pole

5.37.3.9  void FDDlib::DipoleSource2D::setPhases (double phase1, double phase2)

Set the phase angles of both poles.

Parameters:
  phase1  phase of the first pole
  phase2  phase of the second pole

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

- DipoleSource2D.h
5.38  FDDlib::DipoleSource3D Class Reference

Implement a dipole source.

```cpp
#include <DipoleSource3D.h>
```

Inheritance diagram for FDDlib::DipoleSource3D::

```
FDDlib::Source3D
\downarrow
FDDlib::DipoleSource3D
```

**Public Methods**

- `DipoleSource3D` (const `Location3D` &location1, double int1, double phase1, const `Location3D` &location2, double int2, double phase2)
  
  *Constructor.*

- `DipoleSource3D` (const `Location3D` &location1, double int1, double phase1, const `Location3D` &location2, double int2, double phase2, const std::vector<double> &frequencies)
  
  *Constructor.*

- void `set` (const `Location3D` &location1, double int1, double phase1, const `Location3D` &location2, double int2, double phase2)
  
  *Set the attributes of this DipoleSource3D.*

- void `getPoles` (Location3D &location1, double &int1, double &phase1, Location3D &location2, double &int2, double &phase2) const
  
  *Get the poles of this DipoleSource3D by getting all the data directly.*

- void `setLocations` (const `Location3D` &location1, const `Location3D` &location2)
  
  *Set the locations of this dipole source.*

- void `getLocations` (Location3D &location1, Location3D &location2) const
  
  *Get the locations of both poles via reference parameters.*
void getIntensities (double &int1, double &int2) const
Get the intensities of both poles via reference parameters.

void setIntensities (double int1, double int2)
Set the intensities of both poles.

void getPhases (double &phase1, double &phase2) const
Get the phase angles of both poles via reference parameters.

void setPhases (double phase1, double phase2)
Set the phase angles of both poles.

Complex< double > getRightHandSide (const CartesianNode3D &node) const
Get the right-hand-side of the PDE equation focusing on the given node in the grid.

Protected Attributes

- Location3D location1
  location of the first pole

- double int1
  Intensity of first pole.

- double phase1
  Phase of first pole.

- Location3D location2
  location of the second pole

- double int2
  Intensity of second pole.

- double phase2
  Phase of second pole.

5.38.1 Detailed Description

Implement a dipole source.
This is essentially two node sources.
Author:
Kyle Guilbert @change 11/27/02 Removed any interaction with MonopoleSource3D. This functionality was rather useless. @change 12/04/02 Added set functions for phase and intensity.

5.38.2 Constructor & Destructor Documentation

5.38.2.1 FDDlib::DipoleSource3D::DipoleSource3D (const Location3D & location1, double int1, double phase1, const Location3D & location2, double int2, double phase2)

Constructor.

Parameters:
- location1: location of the first pole
- int1: intensity of the first pole
- phase1: phase of the first pole
- location2: location of the second pole
- int2: intensity of the second pole
- phase2: phase of the second pole

5.38.2.2 FDDlib::DipoleSource3D::DipoleSource3D (const Location3D & location1, double int1, double phase1, const Location3D & location2, double int2, double phase2, const std::vector<double> & frequencies)

Constructor.

Parameters:
- location1: location of the first pole
- int1: intensity of the first pole
- phase1: phase of the first pole
- location2: location of the second pole
- int2: intensity of the second pole
- phase2: phase of the second pole
- frequencies: list of modulation frequencies
- numFreq: number of frequencies in the list (range: positive integers)
5.38.3 Member Function Documentation

5.38.3.1 void FDDlib::DipoleSource3D::getIntensities (double & int1, double & int2) const

Get the intensities of both poles via reference parameters.

Parameters:
- int1 intensity of the first pole
- int2 intensity of the second pole

5.38.3.2 void FDDlib::DipoleSource3D::getLocations (Location3D & location1, Location3D & location2) const

Get the locations of both poles via reference parameters.

Parameters:
- location1 location of the first pole
- location2 location of the second pole

5.38.3.3 void FDDlib::DipoleSource3D::getPhases (double & phase1, double & phase2) const

Get the phase angles of both poles via reference parameters.

Parameters:
- phase1 phase of the first pole
- phase2 phase of the second pole

5.38.3.4 void FDDlib::DipoleSource3D::getPoles (Location3D & location1, double & int1, double & phase1, Location3D & location2, double & int2, double & phase2) const

Get the poles of this DipoleSource3D by getting all the data directly.
Parameters:

- location1 location of the first pole
- int1 intensity of the first pole
- phase1 phase of the first pole
- location2 location of the second pole
- int2 intensity of the second pole
- phase2 phase of the second pole

5.38.3.5  Complex<double> FDDlib::DipoleSource3D::getRightHandSide
          (const CartesianNode3D & node) const  [virtual]

Get the right-hand-side of the PDE equation focusing on the given node in the grid.

Parameters:
- node the node of interest (in 3D space)

Returns:
- complex right-hand-side

Implements FDDlib::Source3D.

5.38.3.6  void FDDlib::DipoleSource3D::set (const Location3D & location1,
                                               double int1, double phase1,
                                               const Location3D & location2, double int2,
                                               double phase2)

Set the attributes of this DipoleSource3D.

Parameters:
- location1 location of the first pole
- int1 intensity of the first pole
- phase1 phase of the first pole
- location2 location of the second pole
- int2 intensity of the second pole
- phase2 phase of the second pole
5.38.3.7 void FDDlib::DipoleSource3D::setIntensities (double int1, double int2)

Set the intensities of both poles.

Parameters:
int1 intensity of the first pole
int2 intensity of the second pole

5.38.3.8 void FDDlib::DipoleSource3D::setLocations (const Location3D & location1, const Location3D & location2)

Set the locations of this dipole source.

Parameters:
location1 location of the first pole
location2 location of the second pole

5.38.3.9 void FDDlib::DipoleSource3D::setPhases (double phase1, double phase2)

Set the phase angles of both poles.

Parameters:
phase1 phase of the first pole
phase2 phase of the second pole

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

- DipoleSource3D.h
5.39 FDDlib::DynamicSparseRowMatrix< T > Class Template Reference

Sparse row-column matrix with dynamic growth.
#include <DynamicSparseRowMatrix.h>
Inheritance diagram for FDDlib::DynamicSparseRowMatrix< T >::

```
FDDlib::Matrix< T >

FDDlib::SparseRowMatrix< T >

FDDlib::DynamicSparseRowMatrix< T >
```

Public Methods

- **DynamicSparseRowMatrix ()**
  *constructor*

- **DynamicSparseRowMatrix (int maxrows, int max_per_row)**
  *constructor*

- **DynamicSparseRowMatrix (const DynamicSparseRowMatrix< T > &ESR)**
  *copy constructor*

- int **set (int row, int col, T val)**
  *Set a row and column.*

- int **setEntriesPerRow (int newmax)**
  *Increase the number of entries stored, per row.*

5.39.1 Detailed Description

template<class T> class FDDlib::DynamicSparseRowMatrix< T >

Sparse row-column matrix with dynamic growth.
This class extends the sparse row-order matrix class to allow for some dynamic growth. The set() method will be somewhat slower, but the maximum number of entries stored per row is automatically increased as needed.

**Author:**
Kyle Guilbert @change Modified setHelper() to remove break; 1/15/04 J. Black

### 5.39.2 Member Function Documentation

#### 5.39.2.1 template<class T> int FDDlib::DynamicSparseRowMatrix< T >::setEntriesPerRow (int newmax) [inline]

Increase the number of entries stored, per row.

This function attempts to find new space to store the matrix, and rearranges all the entries to evenly divide the new space amongst the entire entry list.

**Return values:**
- **ERR_ARGS** if invalid newmax is given
- **ERR_DATA** if unable to reallocate
- **SUCCESS** otherwise

**Parameters:**
- **newmax** New number of entries per row

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

- DynamicSparseRowMatrix.h
5.40  **FDDlib::EllipsoidalAnomaly Class Reference**

Acts as an ellipsoidal inhomogeneity in the medium.

```cpp
#include <EllipsoidalAnomaly.h>
```

Inheritance diagram for FDDlib::EllipsoidalAnomaly::

```
FDDlib::Anomaly3D

FDDlib::EllipsoidalAnomaly
```

**Public Methods**

- **EllipsoidalAnomaly** (const `Location3D` &center, double xlen, double ylen, double zlen, double theta1, double theta2, double theta3, `Property *prop`) throw (std::string)  
  *Constructor.*

- **Location3D getCenter** () const  
  *Get the center of the ellipsoid.*

- void **setCenter** (const `Location3D` &center)  
  *Set the center of the ellipsoid.*

- `std::vector<double>` **getLengths** () const  
  *Get the lengths of the semi-axes.*

- void **setLengths** (double xlen, double ylen, double zlen) throw (std::string)  
  *Set the lengths of the semi-axes.*

- `std::vector<double>` **getEulerAngles** () const  
  *Get the angles of the semi-axes.*

- void **setEulerAngles** (double theta1, double theta2, double theta3)  
  *Set the Euler rotation angles.*

- bool **encloses** (const `CartesianNode3D` &node) const
Determine whether this ellipsoid encloses the given node.

**Protected Methods**

- **void updateData ()**
  
  Updates UD^(-2)U^T if U or D change.

**Protected Attributes**

- **Location3D center**
  
  The location (in 3D space) of the ellipsoid's center.

- **double xlen**
  
  length of semi-axis x

- **double ylen**
  
  length of semi-axis y

- **double zlen**
  
  length of semi-axis z

- **double theta1**
  
  Euler angle 1 (x-convention) in radians.

- **double theta2**
  
  Euler angle 2 (x-convention) in radians.

- **double theta3**
  
  Euler angle 3 (x-convention) in radians.

- **DenseMatrix< double > D**
  
  diagonal matrix containing semi-axis lengths

- **DenseMatrix< double > U**
  
  Orthonormal matrix of semi-axis coordinates.

- **DenseMatrix< double > UD^(-2)U^T**
  
  Result of operation U*D^(-2)*transpose(U) to improve performance in encloses().
5.40.1 Detailed Description

Acts as an ellipsoidal inhomogeneity in the medium.

Author:
Kyle Guilbert 5/06/03

5.40.2 Constructor & Destructor Documentation

5.40.2.1 FDDlib::EllipsoidalAnomaly::EllipsoidalAnomaly (const Location3D & center, double xlen, double ylen, double zlen, double theta1, double theta2, double theta3, Property * prop) throw (std::string)

Constructor.

Parameters:
- center  center of the ellipsoid
- xlen length of semi-axis x. Range: (0.0, inf)
- ylen length of semi-axis y. Range: (0.0, inf)
- zlen length of semi-axis z. Range: (0.0, inf)
- theta1 Euler angle 1 in radians, using x-convention
- theta2 Euler angle 2 in radians, using x-convention
- theta3 Euler angle 3 in radians, using x-convention
- prop  Property to use within the ellipsoid

5.40.3 Member Function Documentation

5.40.3.1 bool FDDlib::EllipsoidalAnomaly::encloses (const CartesianNode3D & node) const [virtual]

Determine whether this ellipsoid encloses the given node.

Parameters:
- node  the CartesianNode3D

Return values:
- true  if this ellipsoid encloses the node
false if this ellipsoid does not enclose the node

Implements FDDlib::Anomaly3D.

5.40.3.2 void FDDlib::EllipsoidalAnomaly::setEulerAngles (double \theta_1, double \theta_2, double \theta_3)

Set the Euler rotation angles.
Each is in radians using the x-convention.

Parameters:
- \theta_1 angle 1
- \theta_2 angle 2
- \theta_3 angle 3

5.40.3.3 void FDDlib::EllipsoidalAnomaly::setLengths (double xlen, double ylen, double zlen) throw (std::string)

Set the lengths of the semi-axes.

Parameters:
- xlen length of semi-axis x. Range: (0.0, inf)
- ylen length of semi-axis y. Range: (0.0, inf)
- zlen length of semi-axis z. Range: (0.0, inf)

Exceptions:
- string if any lengths are out of range.

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

- EllipsoidalAnomaly.h
5.41 FDDlib::FiniteDifferences2D Class Reference

Implements the finite differences method.

```cpp
#include <FiniteDifferences2D.h>
```

**Public Methods**

- `FiniteDifferences2D (ArbitraryCartesianGrid2D &acg3d, const std::vector< BoundaryCondition2D *> &bconds)`
  
  *Constructor.*

- `void SingleSrcSolve (int solveIterations, double residualTol, const Source2D &src, const std::vector< Detector2D *> &detarr, Preconditioner &Pre, enum_solver Solver, void(*callback)(const FiniteDifferences2D &, bool, const std::vector< Detector2D *> &)) throw (std::string)`

  *Solve for the solution given a single source.*

- `void MultipleSrcSolve (int solveIterations, double residualTol, const std::vector< Source2D *> &srcarr, const std::vector< Detector2D *> &detarr, Preconditioner &Pre, enum_solver Solver, void(*callback)(const FiniteDifferences2D &, bool, const std::vector< Detector2D *> &)) throw (std::string)`

  *Sequentially solve for the solution field, given a vector of sources having arbitrary frequencies.*

- `bool AdjointSolve (int solveIterations, double residualTol, const Detector2D &adjDetector, double frequency, Preconditioner &Pre, enum_solver Solver) throw (std::string)`

  *Solve for the adjoint field given a Detector2D object that has been used in a direct solve.*

- `bool solve (int solveIterations, double residualTol, bool adjoint, Preconditioner &Pre, RealVector< double > &RHS, enum_solver Solver) throw (std::string)`

  *Solve for the solution field.*

- `ComplexVector< double > getPhi () const`

  *Get the direct solution field.*

- `ComplexVector< double > getAdjointPhi () const`
Get the adjoint field.

- void `getPhi` (RealVector& realvec, RealVector& imagvec) const
  
  Retrieve the direct solution by its real and imaginary components.

- void `getAdjointPhi` (RealVector& realvec, RealVector& imagvec) const
  
  Retrieve the adjoint solution by its real and imaginary components.

- void `getStatistics` (double &solveTime, double &achievedTol, int &usedIterations) const
  
  Get some statistics related to the last PDE solve.

- void `getSourceInfo` (int &srcNum, double &freq) const
  
  Get information about the current source solve.

Protected Methods

- bool `validAndActive` (const s_neighborhood &ndata) const
  
  Returns true if the node exists and is active, given its neighbor data.

- double `getOffDiagVal` (const CartesianNode2D &node, const CartesianNode2D &node_p, const CartesianNode2D &node_m, direction2D dir) const
  
  Get the off-diagonal value to be inserted into the matrix.

- void `resetDataVector` (std::vector<s_neighborhood> &datavec) const
  
  Reset a vector of s_neighborhood structs to default values.

- void `buildAMatrix` (double omega) throw (std::string)
  
  Set up the A matrix using the finite differences method.

- void `postProcessAMatrix` ()
  
  Post-process A by removing the rows that show known values.

- void `updateRightHandSide` (RealVector& RHS)
  
  Update the RHS in accordance with the post-processed A matrix.

- void `copySubMatrices` ()
  
  Copy the upper two matrix blocks to the bottom half of A to create the following form:
  
  \[
  \begin{bmatrix}
  R & I \\
  R & -I
  \end{bmatrix}
  \]
Protected Attributes

- **ArbitraryCartesianGrid2D * grid**  
  pointer to the `ArbitraryCartesianGrid2D`

- **SparseRowMatrix< double > Amat_**  
  The matrix used to solve the partial differential equation.

- **ComplexVector< double > Phi_**  
  Vector containing the direct solution.

- **ComplexVector< double > AdjointPhi_**  
  Vector containing the adjoint solution.

- **int startTime**  
  time (in clock cycles) when timer was started

- **int stopTime**  
  time (in clock cycles) when timer was stopped

- **double achievedTol_**  
  achieved residual tolerance

- **int usedIterations_**  
  number of iterations used

- **int currSrcNum_**  
  number of the current source (starts from 1)

- **double currFreq_**  
  current frequency in use

- **int numUnknowns**  
  number of unknowns in system of equations

- **std::map< int, Complex< double > > knowns**  
  map of known values (the keys are the indices)

- **std::vector< BoundaryCondition2D * > bconds_**  
  the vector of boundary conditions we need to apply

- **SparseRowMatrix< double > Adep**
This sparse matrix stores the coefficients multiplying known values in the original A matrix.

- std::vector<int> rowMap
  This is used to map equations in the system to their position in the reduced system.

### Static Protected Attributes

- const int UNKNOWN_VALUE = -10
  value given to unknown data
- const int OMITVAL = -1
  value given to rows which will not be used in the system

#### 5.41.1 Detailed Description

Implements the finite differences method.

**Author:**
Kyle Guilbert

#### 5.41.2 Constructor & Destructor Documentation

##### 5.41.2.1 FDDlib::FiniteDifferences2D::FiniteDifferences2D

(ArbitraryCartesianGrid2D * acg3d, const std::vector<
  BoundaryCondition2D * > & bconds)

Constructor.

**Parameters:**
- acg3d Pointer to the ArbitraryCartesianGrid2D
- bconds Vector of boundary conditions

#### 5.41.3 Member Function Documentation

Generated on Mon Aug 30 15:41:16 2004 for FDDLib by Doxygen
5.41.3.1 bool FDDlib::FiniteDifferences2D::AdjointSolve (int solveIterations, double residualTol, const Detector2D & adjDetector, double frequency, Preconditioner & Pre, enum_solver Solver) throw (std::string)

Solve for the adjoint field given a Detector2D object that has been used in a direct solve.

Parameters:
- `solveIterations` Maximum of iterations to use in iterative solver. Range: (0, inf)
- `residualTol` Desired tolerance for solution residual. Range: [0, inf)
- `adjDetector` Detector2D object (should have received data already)
- `frequency` Frequency of the modulation
- `Pre` The Preconditioner object
- `Solver` The iterative solver type

Return values:
- `true` if solution residual is less than residualTol
- `false` otherwise

Exceptions:
- `string` on invalid input.

5.41.3.2 void FDDlib::FiniteDifferences2D::buildAMatrix (double omega) throw (std::string) [protected]

Set up the A matrix using the finite differences method.

Parameters:
- `omega` Frequency of the source modulation.

5.41.3.3 void FDDlib::FiniteDifferences2D::getAdjointPhi (RealVector< double > & realvec, RealVector< double > & imagvec) const

Retrieve the adjoint solution by its real and imaginary components.

Parameters:
- `realvec` An empty RealVector for the real part
- `imagvec` An empty RealVector for the imaginary part
5.41.3.4 ComplexVector<

FDDlib::FiniteDifferences2D::getAdjoint-
Phi ()

Get the adjoint field.

**Returns:**
A ComplexVector containing the adjoint solution.

5.41.3.5 double FDDlib::FiniteDifferences2D::getOffDiagVal (const
CartesianNode2D & node, const CartesianNode2D & node.p, const
CartesianNode2D & node.m, direction2D dir) const [protected]

Get the off-diagonal value to be inserted into the matrix.

**Parameters:**
- **node** The node of interest
- **node.p** The node's neighbor in a positive direction
- **node.m** The node's neighbor in a negative direction
- **dir** Which direction we are interested in

5.41.3.6 void FDDlib::FiniteDifferences2D::getPhi (RealVector< double > &
realvec, RealVector< double > & imagvec) const

Retrieve the direct solution by its real and imaginary components.
This is useful for writing the separate components to binary files.

**Parameters:**
- **realvec** An empty RealVector for the real part
- **imagvec** An empty RealVector for the imaginary part

5.41.3.7 ComplexVector<

FDDlib::FiniteDifferences2D::getPhi ()

Get the direct solution field.
A new vector is allocated for this.
Returns:
A double-precision ComplexVector containing the solution.

5.41.3.8 void FDDlib::FiniteDifferences2D::getSourceInfo (int & srcNum, double & freq) const

Get information about the current source solve.

Parameters:
srcNum  Number of the current source in the source list. For a single source solve, this will always be one. For a multiple source solve, this starts from one and is updated after every solve.

freq  The current frequency.

Note: This is a utility function, intended for use in the callback function used in Multiple-SrcSolve.

5.41.3.9 void FDDlib::FiniteDifferences2D::getStatistics (double & solveTime, double & achievedTol, int & usedIterations) const

Get some statistics related to the last PDE solve.
This is done with reference arguments.

Parameters:
solveTime  Time in seconds it took to complete the solve.

achievedTol  The achieved residual tolerance.

usedIterations  Number of iterations used.

5.41.3.10 void FDDlib::FiniteDifferences2D::MultipleSrcSolve (int solveIterations, double residualTol, const std::vector< Source2D * > & srcarr, const std::vector< Detector2D * > & detarr, Preconditioner & Pre, enum_solver Solver, void(* callback)(const FiniteDifferences2D &, bool, const std::vector< Detector2D * > &)) throw (std::string)

Sequentially solve for the solution field, given a vector of sources having arbitrary frequencies.
The A matrix is rebuilt for each Source2D to account for changes in frequency. To be able to extract the solution after each of these solves, we utilize a function pointer.

**Parameters:**

- **solveIterations**  Maximum of iterations to use in iterative solver. Range: (0, inf)
- **residualTol** Desired tolerance for solution residual. Range: [0, inf)
- **srcarr** Vector of Source2D pointers
- **detarr** Vector of Detector2D pointers
- **Pre**  The Preconditioner object
- **Solver** The iterative solver type
- **callback** A function pointer. The function pointed to must have the following parameters: (const FiniteDifferences2D&, bool, const std::vector<Detector2D*>&). The second boolean parameter is true if the residual was less than residualTol and false otherwise.

**Exceptions:**

- **string**  on invalid input.

5.41.3.11 void FDDlib::FiniteDifferences2D::resetDataVector (std::vector<s_neighbordata> & datavec) const

Reset a vector of s_neighbordata structs to default values.

**Parameters:**

- **datavec**  Vector of s_neighbordata structs

5.41.3.12 void FDDlib::FiniteDifferences2D::SingleSrcSolve (int solveIterations, double residualTol, const Source2D & src, const std::vector<Detector2D*>& detarr, Preconditioner & Pre, enum_solver Solver, void(*callback)(const FiniteDifferences2D & src, bool, const std::vector<Detector2D*>&)) throw (std::string)

Solve for the solution given a single source.

Since this source may operate at a range of frequencies, a callback function is used to allow access to the solution data after each solve.

**Parameters:**

- **solveIterations** Maximum of iterations to use in iterative solver. Range: (0, inf)
**residualTol**  Desired tolerance for solution residual. Range: [0, inf)

**src**  The Source2D pointer

**detarr**  Vector of Detector2D pointers

**Pre**  The Preconditioner object

**Solver**  The iterative solver type

**callback**  A function pointer. The function pointed to must have the following parameters: (const FiniteDifferences2D&, bool, const std::vector<Detector2D*>&). The second boolean parameter is true if the residual was less than residualTol and false otherwise.

**Exceptions:**

- **string**  on invalid input.

### 5.41.3.13  bool FDDlib::FiniteDifferences2D::solve (int solveIterations, double residualTol, bool adjoint, Preconditioner & Pre, RealVector< double > & RHS, enum_solver Solver) throw (std::string)

Solve for the solution field.

This is where the IML solver routines are actually called.

**Parameters:**

- **solveIterations**  Maximum of iterations to use in iterative solver. Range: (0, inf)
- **residualTol**  Desired tolerance for solution residual. Range: [0, inf)
- **adjoint**  Whether we are solving for the adjoint problem. (true for adjoint, false for direct)
- **Pre**  Preconditioner object
- **RHS**  The right-hand-side of the equation (source vector)
- **Solver**  The iterative solver type

**Note:**

This function assumes the A matrix is already built.

**Exceptions:**

- **string**  on invalid input.  @bugfix 11/12/02 Zeros in off-diagonal positions no longer produce an error.
5.41.3.14 bool FDDlib::FiniteDifferences2D::validAndActive (const s_neighbordata & ndata) const [protected]

Returns true if the node exists and is active, given its neighbor data.

Parameters:
   ndata Neighbor data of the node

5.41.4 Member Data Documentation

5.41.4.1 std::vector<int> FDDlib::FiniteDifferences2D::rowMap [protected]

This is used to map equations in the system to their position in the reduced system. Rows not in the reduced system have a -1 in their position.

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

- FiniteDifferences2D.h
5.42 FDDlib::FiniteDifferences2D::s_neighbordata Struct Reference

Structure used to store info about a node’s neighbors.

#include <FiniteDifferences2D.h>

Public Attributes

- CartesianNode2D * n_node
  
  The neighboring node.

- int n_linearIndex
  
  The neighboring node’s linear index.

5.42.1 Detailed Description

Structure used to store info about a node’s neighbors.

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

- FiniteDifferences2D.h
5.43  FDDlib::FiniteDifferences3D Class Reference

Implements the finite differences method.

#include <FiniteDifferences3D.h>

Public Methods

- **FiniteDifferences3D** (ArbitraryCartesianGrid3D *acg3d, const std::vector<
  BoundaryCondition3D *> &bconds)
  Constructor.

- void **SingleSrcSolve** (int solveIterations, double residualTol, const 
  Source3D &src, const std::vector< 
  Detector3D *> &detarr, Preconditioner &Pre, 
  enum_solver Solver, void(*callback)(const FiniteDifferences3D &, 
  bool, const std::vector< 
  Detector3D *> &)) throw (std::string)
  Solve for the solution given a single source.

- void **MultipleSrcSolve** (int solveIterations, double residualTol, const 
  std::vector<
  Source3D *> &srcarr, const std::vector<
  Detector3D *> &detarr, Preconditioner &Pre, 
  enum_solver Solver, void(*callback)(const FiniteDifferences3D &, 
  bool, const std::vector<
  Detector3D *> &)) throw (std::string)
  Sequentially solve for the solution field, given a vector of sources having 
  arbitrary frequencies.

- bool **AdjointSolve** (int solveIterations, double residualTol, const 
  Detector3D &adjDetector, double frequency, 
  Preconditioner &Pre, enum_solver Solver) throw (std::string)
  Solve for the adjoint field given a Detector3D object that has been used in a 
  direct solve.

- bool **solve** (int solveIterations, double residualTol, 
  bool adjoint, Preconditioner &Pre, 
  RealVector< double > &RHS, enum_solver Solver) throw (std::string)
  Solve for the solution field.

- ComplexVector< double > **getPhi** () const
  Get the direct solution field.

- ComplexVector< double > **getAdjointPhi** () const
Get the adjoint field.

- void getPhi (RealVector< double > &realvec, RealVector< double > &imagvec) const
  
  Retrieve the direct solution by its real and imaginary components.

- void getAdjointPhi (RealVector< double > &realvec, RealVector< double > &imagvec) const
  
  Retrieve the adjoint solution by its real and imaginary components.

- void getStatistics (double &solveTime, double &achievedTol, int &usedIterations) const
  
  Get some statistics related to the last PDE solve.

- void getSourceInfo (int &srcNum, double &freq) const
  
  Get information about the current source solve.

Protected Methods

- bool validAndActive (const s_neighbordata &ndata) const
  
  Returns true if the node exists and is active, given its neighbor data.

- double getOffDiagVal (const CartesianNode3D &node, const CartesianNode3D &nodeP, const CartesianNode3D &nodeM, direction3D dir) const
  
  Get the off-diagonal value to be inserted into the matrix.

- void resetDataVector (std::vector< s_neighbordata > &datavec) const
  
  Reset a vector of s_neighbordata structs to default values.

- void buildAMatrix (double omega) throw (std::string)
  
  Set up the A matrix using the finite differences method.

- void postProcessAMatrix ()
  
  Post-process A by removing the rows that show known values.

- void updateRightHandSide (RealVector< double > &RHS)
  
  Update the RHS in accordance with the post-processed A matrix.

- void copySubMatrices ()
  
  Copy the upper two matrix blocks to the bottom half of A to create the following form:
  \[ \begin{bmatrix} R & I \\ R & -I \end{bmatrix} \].
Protected Attributes

- `ArbitraryCartesianGrid3D * grid`
  pointer to the `ArbitraryCartesianGrid3D`

- `SparseRowMatrix< double > Amat`
  The matrix used to solve the partial differential equation.

- `ComplexVector< double > Phi`
  Vector containing the direct solution.

- `ComplexVector< double > AdjointPhi`
  Vector containing the adjoint solution.

- `int startTime`
  time (in clock cycles) when timer was started

- `int stopTime`
  time (in clock cycles) when timer was stopped

- `double achievedTol`
  achieved residual tolerance

- `int usedIterations`
  number of iterations used

- `int currSrcNum`
  number of the current source (starts from 1)

- `double currFreq`
  current frequency in use

- `int numUnknowns`
  number of unknowns in system of equations

- `std::map< int, Complex< double > > knowns`
  map of known values (the keys are the indeces)

- `std::vector< BoundaryCondition3D * > bconds`
  the vector of boundary conditions we need to apply

- `SparseRowMatrix< double > Adep`
This sparse matrix stores the coefficients multiplying known values in the _original_ A matrix.

- std::vector<int> rowMap
  This is used to map equations in the system to their position in the reduced system.

**Static Protected Attributes**

- const int UNKNOWN_VALUE = -10
  value given to unknown data

- const int OMITVAL = -1
  value given to rows which will not be used in the system

### 5.43.1 Detailed Description

Implements the finite differences method.

**Author:**
Kyle Guilbert

### 5.43.2 Constructor & Destructor Documentation

#### 5.43.2.1 FDDlib::FiniteDifferences3D::FiniteDifferences3D

(ArbitraryCartesianGrid3D * acg3d, const std::vector<BoundaryCondition3D * > & bconds)

Constructor.

**Parameters:**
- *acg3d* Pointer to the ArbitraryCartesianGrid3D
- *bconds* Vector of boundary conditions

### 5.43.3 Member Function Documentation
5.43.3.1  bool FDDlib::FiniteDifferences3D::AdjointSolve (int solveIterations, double residualTol, const Detector3D & adjDetector, double frequency, Preconditioner & Pre, enum_solver Solver) throw (std::string)

Solve for the adjoint field given a Detector3D object that has been used in a direct solve.

Parameters:
- **solveIterations**  Maximum of iterations to use in iterative solver. Range: (0, inf)
- **residualTol**  Desired tolerance for solution residual. Range: [0, inf)
- **adjDetector3D**  Detector3D object (should have received data already)
- **frequency**  Frequency of the modulation
- **Pre**  The Preconditioner object
- **Solver**  The iterative solver type

Return values:
- **true**  if solution residual is less than residualTol
- **false**  otherwise

Exceptions:
- **string**  on invalid input.

5.43.3.2  void FDDlib::FiniteDifferences3D::buildAMatrix (double omega) throw (std::string)  [protected]

Set up the A matrix using the finite differences method.

Parameters:
- **omega**  Frequency of the source modulation.

5.43.3.3  void FDDlib::FiniteDifferences3D::getAdjointPhi (RealVector< double > & realvec, RealVector< double > & imagvec) const

Retrieve the adjoint solution by its real and imaginary components.

Parameters:
- **realvec**  An empty RealVector for the real part
- **imagvec**  An empty RealVector for the imaginary part
5.43.3.4  ComplexVector<double> FDDlib::FiniteDifferences3D::getAdjointPhi ()

Get the adjoint field.

Returns:
A ComplexVector containing the adjoint solution.

5.43.3.5  double FDDlib::FiniteDifferences3D::getOffDiagVal (const CartesianNode3D & node, const CartesianNode3D & node_p, const CartesianNode3D & node_m, direction3D dir) const [protected]

Get the off-diagonal value to be inserted into the matrix.

Parameters:
node  The node of interest
node_p The node’s neighbor in a positive direction
node_m The node’s neighbor in a negative direction
dir  Which direction we are interested in

5.43.3.6  void FDDlib::FiniteDifferences3D::getPhi (RealVector<double> & realvec, RealVector<double> & imagvec) const

Retrieve the direct solution by its real and imaginary components.
This is useful for writing the separate components to binary files.

Parameters:
realvec  An empty RealVector for the real part
imagvec  An empty RealVector for the imaginary part

5.43.3.7  ComplexVector<double> FDDlib::FiniteDifferences3D::getPhi ()

Get the direct solution field.
A new vector is allocated for this.
5.43 FDDlib::FiniteDifferences3D Class Reference

Returns:
A double-precision ComplexVector containing the solution.

5.43.3.8 void FDDlib::FiniteDifferences3D::getSourceInfo (int & srcNum, double & freq) const

Get information about the current source solve.

Parameters:
srcNum Number of the current source in the source list. For a single source solve,
this will always be one. For a multiple source solve, this starts from one and
is updated after every solve.

freq The current frequency.

Note:
This is a utility function, intended for use in the callback function used in Multiple-
SrcSolve.

5.43.3.9 void FDDlib::FiniteDifferences3D::getStatistics (double & solveTime, double & achievedTol, int & usedIterations) const

Get some statistics related to the last PDE solve.
This is done with reference arguments.

Parameters:
solveTime Time in seconds it took to complete the solve.
achievedTol The achieved residual tolerance.
usedIterations Number of iterations used.

5.43.3.10 void FDDlib::FiniteDifferences3D::MultipleSrcSolve (int solveIterations, double residualTol, const std::vector< Source3D * > & srcarr, const std::vector< Detector3D * > & detarr, Preconditioner & Pre, enum solver Solver, void(* callback)(const FiniteDifferences3D &, bool, const std::vector< Detector3D * > &)) throw (std::string)

Sequentially solve for the solution field, given a vector of sources having arbitrary
frequencies.
The A matrix is rebuilt for each Source3D to account for changes in frequency. To be able to extract the solution after each of these solves, we utilize a function pointer.

**Parameters:**

- **solveIterations**  Maximum of iterations to use in iterative solver. Range: (0, inf)
- **residualTol**  Desired tolerance for solution residual. Range: [0, inf)
- **srcarr**  Vector of Source3D pointers
- **detarr**  Vector of Detector3D pointers
- **Pre**  The Preconditioner object
- **Solver**  The iterative solver type
- **callback**  A function pointer. The function pointed to must have the following parameters: (const FiniteDifferences3D&, bool, const std::vector<Detector3D*>&). The second boolean parameter is true if the residual was less than residualTol and false otherwise.

**Exceptions:**

- **string**  on invalid input.

---

**5.43.3.11 void FDDlib::FiniteDifferences3D::resetDataVector (std::vector<s_neighbordata> & datavec) const [protected]**

Reset a vector of s_neighbordata structs to default values.

**Parameters:**

- **datavec**  Vector of s_neighbordata structs

---

**5.43.3.12 void FDDlib::FiniteDifferences3D::SingleSrcSolve (int solveIterations, double residualTol, const Source3D & src, const std::vector<Detector3D*> & detarr, Preconditioner & Pre, enum solver Solver, void(* callback)(const FiniteDifferences3D&, bool, const std::vector<Detector3D*>&)) throw (std::string)**

Solve for the solution given a single source.

Since this source may operate at a range of frequencies, a callback function is used to allow access to the solution data after each solve.

**Parameters:**

- **solveIterations**  Maximum of iterations to use in iterative solver. Range: (0, inf)
residualTol  Desired tolerance for solution residual. Range: [0, inf)
src  The Source3D pointer
detarr  Vector of Detector3D pointers
Pre  The Preconditioner object
Solver  The iterative solver type
callback  A function pointer. The function pointed to must have the following parameters: (const FiniteDifferences3D&, bool, const std::vector<Detector3D*>&). The second boolean parameter is true if the residual was less than residualTol and false otherwise.

Exceptions:
string  on invalid input.

5.43.3.13  bool FDDlib::FiniteDifferences3D::solve (int solveIterations, double residualTol, bool adjoint, Preconditioner & Pre, RealVector< double > & RHS, enum_solver Solver) throw (std::string)

Solve for the solution field.
This is where the IML solver routines are actually called.

Parameters:
solveIterations  Maximum of iterations to use in iterative solver. Range: (0, inf)
residualTol  Desired tolerance for solution residual. Range: [0, inf)
adjoint  Whether we are solving for the adjoint problem. (true for adjoint, false for direct)
Pre  Preconditioner object
RHS  The right-hand-side of the equation (source vector)
Solver  The iterative solver type

Note:
This function assumes the A matrix is already built.

Exceptions:
string  on invalid input. @bugfix 11/12/02 Zeros in off-diagonal positions no longer produce an error.
5.43.3.14 bool FDDlib::FiniteDifferences3D::validAndActive (const s_neighbordata & ndata) const [protected]

Returns true if the node exists and is active, given its neighbor data.

Parameters:

   ndata  Neighbor data of the node

5.43.4 Member Data Documentation

5.43.4.1 std::vector<int> FDDlib::FiniteDifferences3D::rowMap [protected]

This is used to map equations in the system to their position in the reduced system. Rows not in the reduced system have a -1 in their position.

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

- FiniteDifferences3D.h
5.44 FDDlib::FiniteDifferences3D::s_neighbordata Struct Reference

Structure used to store info about a node’s neighbors.

#include <FiniteDifferences3D.h>

Public Attributes

- CartesianNode3D * n_node
  
  The neighboring node.

- int n_linearIndex
  
  The neighboring node’s linear index.

5.44.1 Detailed Description

Structure used to store info about a node’s neighbors.

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

- FiniteDifferences3D.h
5.45 FDDlib::ILU0Preconditioner Class Reference

Implement an ILU(0) preconditioner.

#include <ILU0Preconditioner.h>

Inheritance diagram for FDDlib::ILU0Preconditioner:

```
FDDlib::Preconditioner
  ↓
FDDlib::ILU0Preconditioner
```

Public Methods

- **ILU0Preconditioner ()**
  constructor

- **~ILU0Preconditioner ()**
  destructor

- void **init (SparseRowMatrix< double > &srmd)** throw (std::string)
  initialize preconditioner data

- template<class Vector> Vector **lowerSolve (const Vector &x)** const
  Solve the lower triangular system: \( Lx = y \).

- template<class Vector> Vector **upperSolve (const Vector &x)** const
  Solve the upper-triangular system: \( Uy = x \).

- RealVector< double > **solve (const RealVector< double > &x)** const
  Solve the system by doing a lower solve followed by an upper solve.

5.45.1 Detailed Description

Implement an ILU(0) preconditioner.

Has the same zero pattern as the original matrix.
5.45 FDDlib::ILU0Preconditioner Class Reference

Author:
Greg Boverman 2-8-02

5.45.2 Member Function Documentation

5.45.2.1 void FDDlib::ILU0Preconditioner::init (SparseRowMatrix< double > & srmd) throw (std::string) [inline, virtual]
initialize preconditioner data

Parameters:
  srmd  Our extensible sparse row matrix

Implements FDDlib::Preconditioner.

5.45.2.2 template<class Vector> Vector FDDlib::ILU0Preconditioner::lower-Solve (const Vector & x) const [inline]
Solve the lower triangular system: Lx = y.

Parameters:
  x  right hand side

Return values:
  the  solution vector

5.45.2.3 RealVector< double > FDDlib::ILU0Preconditioner::solve (const RealVector< double > & x) const [inline, virtual]
Solve the system by doing a lower solve followed by an upper solve.

Parameters:
  x  right hand side

Return values:
  the  solution vector

Implements FDDlib::Preconditioner.
5.45.2.4 template<class Vector> Vector FDDlib::ILU0Preconditioner::upper-Solve (const Vector & x) const  [inline]

Solve the upper-triangular system: Uy = x.

Parameters:

  x  right hand side

Return values:

  the  solution vector

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

  • ILU0Preconditioner.h
A class to store an absolute location in 2-dimensional space.

#include <Location2D.h>

Public Methods

- Location2D ()
  Constructor.

- Location2D (double xloc, double yloc)
  Constructor.

- void setCoordinates (double xloc, double yloc)
  Set the coordinates.

- Location2D operator- (const Location2D &loc) const
  Subtract another location from this one, returning the result.

- bool operator== (const Location2D &loc) const
  Compare this location with another.

Public Attributes

- double x
  X coordinate.

- double y
  Y coordinate.

5.46.1 Detailed Description

A class to store an absolute location in 2-dimensional space.
Since this is a very basic and widely used data structure, we make location modification and retrieval simple by using public data.
Author:
Kyle Guilbert 2/25/03

5.46.2 Constructor & Destructor Documentation

5.46.2.1 FDDlib::Location2D::Location2D (double xloc, double yloc)

Constructor.

Parameters:
  xloc  X coordinate
  yloc  Y coordinate

5.46.3 Member Function Documentation

5.46.3.1 Location2D FDDlib::Location2D::operator- (const Location2D & loc)

Subtract another location from this one, returning the result.

Parameters:
  loc  Another 2D location

5.46.3.2 bool FDDlib::Location2D::operator== (const Location2D & loc)

Compare this location with another.
We account for floating-point representation error here.

Parameters:
  loc  another 2D location.
5.46.3.3  void FDDlib::Location2D::setCoordinates (double xloc, double yloc)

Set the coordinates.

Parameters:
   xloc  X coordinate
   yloc  Y coordinate

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

- Location2D.h
5.47 FDDlib::Location3D Class Reference

A class to store an absolute location in 3-dimensional space.
#include <Location3D.h>

Public Methods

- Location3D ()
  Constructor.

- Location3D (double xloc, double yloc, double zloc)
  Constructor.

- void setCoordinates (double xloc, double yloc, double zloc)
  Set the coordinates.

- Location3D operator- (const Location3D &loc) const
  Subtract another location from this one, returning the result.

- bool operator== (const Location3D &loc) const
  Compare this location with another.

Public Attributes

- double x
  X location.

- double y
  Y location.

- double z
  Z location.
5.47 FDDlib::Location3D Class Reference

5.47.1 Detailed Description

A class to store an absolute location in 3-dimensional space.
Since this is a very basic and widely used data structure, we make location modification
and retrieval simple by using public data.

Author: Kyle Guilbert 2/25/03

5.47.2 Constructor & Destructor Documentation

5.47.2.1 FDDlib::Location3D::Location3D (double xloc, double yloc, double zloc)
Constructor.

Parameters:
  xloc X coordinate
  yloc Y coordinate
  zloc Z coordinate

5.47.3 Member Function Documentation

5.47.3.1 Location3D FDDlib::Location3D::operator- (const Location3D & loc) const
Subtract another location from this one, returning the result.

Parameters:
  loc Another 3D location

5.47.3.2 bool FDDlib::Location3D::operator== (const Location3D & loc) const
Compare this location with another.
We account for floating-point representation error here.

Generated on Mon Aug 30 15:41:16 2004 for FDDLib by Doxygen
Parameters:
   \textit{loc} Another 3D location.

5.47.3.3 \texttt{void FDDlib::Location3D::setCoordinates (double xloc, double yloc, double zloc)}

Set the coordinates.

Parameters:
   \textit{xloc} X coordinate
   \textit{yloc} Y coordinate
   \textit{zloc} Z coordinate

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

- Location3D.h
5.48 FDDlib::Matrix< T > Class Template Reference

Base class for matrix operations.
#include <Matrix.h>

Inheritance diagram for FDDlib::Matrix< T >:

```
FDDlib::Matrix< T >
```
```
FDDlib::DenseMatrix< T >
```
```
FDDlib::SparseRowMatrix< T >
```
```
FDDlib::DynamicSparseRowMatrix< T >
```

Public Methods

- virtual void **deallocate** ()=0
  Deallocate this matrix.

- virtual int **getNumRows** () const=0
  get the number of rows in this matrix

- virtual int **getNumCols** () const=0
  get the number of columns in this matrix

- virtual **RealVector< T >** **getRow** (int row) const=0
  get an entire row of the matrix

- virtual **RealVector< T >** **getColumn** (int col) const=0
  get an entire column of the matrix

- virtual T **val** (int row, int col) const=0
  get the value at a row and column

- virtual void **set** (int row, int col, T v)=0 throw (std::string)
  set the value at a row and column.
5.48.1 Detailed Description

template<class T> class FDDlib::Matrix<T>

Base class for matrix operations.

Author:
Greg Boverman 10-1-01, Kyle Guilbert

5.48.2 Member Function Documentation

5.48.2.1 template<class T> virtual void FDDlib::Matrix<T>::set (int row, int col, T v) throw (std::string) [pure virtual]

set the value at a row and column.

Parameters:
 row matrix row
 col matrix column
 v value to assign this element

Exceptions:
 string if unable to perform set

Implemented in FDDlib::DenseMatrix<T>, FDDlib::DynamicSparseRowMatrix<T>, FDDlib::SparseRowMatrix<T>, FDDlib::DenseMatrix<double>, and FDDlib::SparseRowMatrix<double>.

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

- Matrix.h
5.49 FDDlib::MonopoleDetector2D Class Reference

A detector collecting data from a single node.

#include <MonopoleDetector2D.h>

Inheritance diagram for FDDlib::MonopoleDetector2D::

```
FDDlib::Detector2D
   |
FDDlib::MonopoleDetector2D
```

Public Methods

- **MonopoleDetector2D** (const Location2D &location)
  
  *Constructor.*

- **void setLocation** (const Location2D &location)
  
  *Set detector location.*

- **Location2D getLocation** () const
  
  *Get detector location.*

- **void setData** (const CartesianNode2D &node, const Complex<double> &data)
  
  *Set the data at this detector.*

- **Complex<double> getData** () const
  
  *Get the data stored at this detector.*

- **Complex<double> getData** (const CartesianNode2D &node) const
  
  *Get the data stored at this detector, given a node.*

Protected Attributes

- **Location2D location_**
5.49.1 Detailed Description

A detector collecting data from a single node.

Author:
Kyle Guilbert

5.49.2 Constructor & Destructor Documentation

5.49.2.1 FDDlib::MonopoleDetector2D::MonopoleDetector2D (const Location2D & location)

Constructor.

Parameters:
location The location of this detector

5.49.3 Member Function Documentation

5.49.3.1 Complex<double> FDDlib::MonopoleDetector2D::getData (const CartesianNode2D & node) const [virtual]

Get the data stored at this detector, given a node.
Here we check to see if our coordinates match the node’s coordinates.

Parameters:
node The node of interest (in 2D space)

Implements FDDlib::Detector2D.
5.49.3.2  void FDDlib::MonopoleDetector2D::setData (const CartesianNode2D & node, const Complex & double & data) [virtual]

Set the data at this detector.
We simply check to see if the given node has the same location as this detector, and set the data accordingly.

Parameters:
   node The node, in 2D space, where the data exists
   data The complex data @bugfix 11/12/02 Now accounts for floating-node representation error

Implements FDDlib::Detector2D.

5.49.3.3  void FDDlib::MonopoleDetector2D::setLocation (const Location2D & location)

Set detector location.

Parameters:
   location The location of this detector

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

- MonopoleDetector2D.h
5.50  FDDlib::MonopoleDetector3D Class Reference

A detector collecting data from a single node.

```cpp
#include <MonopoleDetector3D.h>
```

Inheritance diagram for FDDlib::MonopoleDetector3D::

```
FDDlib::Detector3D
   |
   V
FDDlib::MonopoleDetector3D
```

**Public Methods**

- **MonopoleDetector3D** (const `Location3D` &location)
  
  Constructor.

- **void setLocation** (const `Location3D` &location)
  
  Set detector location.

- **Location3D getLocation () const**
  
  Get detector location.

- **void setData** (const `CartesianNode3D` &node, const `Complex< double >` &data)
  
  Set the data at this detector.

- **Complex< double > getData () const**
  
  Get the data stored at this detector.

- **Complex< double > getData (const `CartesianNode3D` &node) const**
  
  Get the data stored at this detector, given a node.

**Protected Attributes**

- **Location3D location**
5.50 FDDlib::MonopoleDetector3D Class Reference

location of this detector in 3D space

- Complex< double > data_
  the solution at this detector

5.50.1 Detailed Description

A detector collecting data from a single node.

Author:
  Kyle Guilbert

5.50.2 Constructor & Destructor Documentation

5.50.2.1 FDDlib::MonopoleDetector3D::MonopoleDetector3D (const Location3D & location)

Constructor.

Parameters:
  location The location of this detector

5.50.3 Member Function Documentation

5.50.3.1 Complex< double > FDDlib::MonopoleDetector3D::getData (const CartesianNode3D & node) const [virtual]

Get the data stored at this detector, given a node.

Here we check to see if our coordinates match the node’s coordinates.

Parameters:
  node The node of interest (in 3D space)

Implements FDDlib::Detector3D.
5.50.3.2  void FDDlib::MonopoleDetector3D::setData (const CartesianNode3D & node, const Complex< double > & data)  [virtual]

Set the data at this detector.

We simply check to see if the given node has the same location as this detector, and set the data accordingly.

**Parameters:**

- **node**  The node, in 3D space, where the data exists
- **data**  The complex data @bugfix 11/12/02 Now accounts for floating-node representation error

Implements FDDlib::Detector3D.

5.50.3.3  void FDDlib::MonopoleDetector3D::setLocation (const Location3D & location)

Set detector location.

**Parameters:**

- **location**  The location of this detector

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

- MonopoleDetector3D.h
Implement a monopole source.

```cpp
#include <MonopoleSource2D.h>
```

Inheritance diagram for FDDlib::MonopoleSource2D:

```
FDDlib::Source2D
```

```
FDDlib::MonopoleSource2D
```

### Public Methods

- **MonopoleSource2D** (const `Location2D` &location, double intensity, double phase)
  
  Constructor.

- **MonopoleSource2D** (const `Location2D` &location, double intensity, double phase, const `std::vector<double>` &frequencies)
  
  Constructor.

- **void set** (const `Location2D` &location, double intensity, double phase)
  
  Set the attributes of this `MonopoleSource2D`.

- **void setLocation** (const `Location2D` &location)
  
  Set location of source.

- **`Location2D` getLocation () const**
  
  Get the source location.

- **void setPhase** (double phase)
  
  Set the phase.

- **void setIntensity** (double intensity)
  
  Set the intensity.

- **double getPhase () const**
Get the phase.

- double **getIntensity** () const
  
  Get the intensity.

- **Complex< double > getRightHandSide** (const **CartesianNode2D** &node) const
  
  Get the right-hand-side of the PDE equation focusing on the given node in the grid.

### Protected Attributes

- **Location2D location_**
  
  the location of this source in 2D space

- double **intensity_**
  
  intensity of energy in this source

- double **phase_**
  
  phase of source

### 5.51.1 Detailed Description

Implement a monopole source.
Stores the location of the source as well as its intensity, phase, and frequency.

**Author:**
Kyle Guilbert

### 5.51.2 Constructor & Destructor Documentation

#### 5.51.2.1 FDDlib::MonopoleSource2D::MonopoleSource2D (const Location2D &location, double intensity, double phase)

Constructor.

**Parameters:**

- **location** The location of this source
5.51 FDDlib::MonopoleSource2D Class Reference

5.51.2.2 FDDlib::MonopoleSource2D::MonopoleSource2D (const Location2D & location, double intensity, double phase, const std::vector<double> & frequencies)

Constructor.

Parameters:
- location The location of this source
- intensity The source intensity
- phase The source phase
- frequencies List of modulation frequencies

5.51.3 Member Function Documentation

5.51.3.1 Complex<double> FDDlib::MonopoleSource2D::getRightHandSide (const CartesianNode2D & node) const [virtual]

Get the right-hand-side of the PDE equation focusing on the given node in the grid.

Parameters:
- node The node of interest (in 2D space)

Returns:
- complex right-hand-side

Implements FDDlib::Source2D.

5.51.3.2 void FDDlib::MonopoleSource2D::set (const Location2D & location, double intensity, double phase)

Set the attributes of this MonopoleSource2D.

Parameters:
- location The location of this source
intensity  The source intensity
phase    The source phase

5.51.3.3  void FDDlib::MonopoleSource2D::setLocation (const Location2D & location)
Set location of source.
Parameters:
  location  The location of this source

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

- MonopoleSource2D.h
5.52 FDDlib::MonopoleSource3D Class Reference

Implement a monopole source.

`#include <MonopoleSource3D.h>`

Inheritance diagram for FDDlib::MonopoleSource3D:

```
FDDlib::Source3D

FDDlib::MonopoleSource3D
```

Public Methods

- `MonopoleSource3D` (const `Location3D` &location, double intensity, double phase)
  
  Constructor.

- `MonopoleSource3D` (const `Location3D` &location, double intensity, double phase, const std::vector<double>& frequencies)
  
  Constructor.

- void `set` (const `Location3D` &location, double intensity, double phase)
  
  Set the attributes of this `MonopoleSource3D`.

- void `setLocation` (const `Location3D` &location)
  
  Set location of source.

- `Location3D` `getLocation` () const
  
  Get the source location.

- void `setPhase` (double phase)
  
  Set the phase.

- void `setIntensity` (double intensity)
  
  Set the intensity.

- double `getPhase` () const
Get the phase.

- double getIntensity() const
  Get the intensity.

- Complex< double > getRightHandSide(const CartesianNode3D &node) const
  Get the right-hand-side of the PDE equation focusing on the given node in the grid.

Protected Attributes

- Location3D location_
  the location of this source in 3D space

- double intensity_
  intensity of energy in this source

- double phase_
  phase of source

5.52.1 Detailed Description

Implement a monopole source.
Stores the location of the source as well as its intensity, phase, and frequency.

Author:
Kyle Guilbert

5.52.2 Constructor & Destructor Documentation

5.52.2.1 FDDlib::MonopoleSource3D::MonopoleSource3D (const Location3D &location, double intensity, double phase)

Constructor.

Parameters:
  location The location of this source
5.52 FDDlib::MonopoleSource3D Class Reference

The source intensity

phase The source phase

5.52.2.2 FDDlib::MonopoleSource3D::MonopoleSource3D (const Location3D & location, double intensity, double phase, const std::vector<double> & frequencies)

Constructor.

Parameters:
location The location of this source
intensity The source intensity
phase The source phase
frequencies List of modulation frequencies

5.52.3 Member Function Documentation

5.52.3.1 Complex<double> FDDlib::MonopoleSource3D::getRightHandSide (const CartesianNode3D & node) const [virtual]

Get the right-hand-side of the PDE equation focusing on the given node in the grid.

Parameters:
node The node of interest (in 3D space)

Returns:
complex right-hand-side

Implements FDDlib::Source3D.

5.52.3.2 void FDDlib::MonopoleSource3D::set (const Location3D & location, double intensity, double phase)

Set the attributes of this MonopoleSource3D.

Parameters:
location The location of this source
**intensity** The source intensity

**phase** The source phase

### 5.52.3.3 void FDDlib::MonopoleSource3D::setLocation (const Location3D & location)

Set location of source.

**Parameters:**
- **location** The location of this source

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

- MonopoleSource3D.h
A class in which a configuration file can be defined and, once defined, traversed through.

```
#include <Parser.h>
```

### Public Types

- **enum Dimension**
  
  Enumeration for dimensionality.

- **enum BoundType**
  
  Enumeration for Boundary types.

- **enum PreconType**
  
  Enumeration for Preconditioner types.

- **enum BackgroundType**
  
  Enumeration for property-background types.

- **enum GridType**
  
  Enumeration for grid types.

- **enum TagType**
  
  Enumerations for each tag type.

### Public Methods

- **Parser (char *path, Dimension d) throw (std::string)**
  
  Constructor.

- **void parseFile () throw (std::string)**
  
  Parse the file.

- **Dimension getDomain () const**
  
  Get the dimensionality (i.e. 2D/3D) of the problem.
- `std::string getFilename() const`  
  Get the configuration file path.

- `bool querySuccessfulParse() const`  
  Returns whether the parsing was successful.

- `GridType getGridType() const`  
  Get the type of grid we are using.

- `s_arbcartgrid getArbitraryCartesianGridData() const throw (std::string)`  
  Get the structure containing arbitrary cartesian grid data.

- `s_regcartgrid getRegularCartesianGridData() const throw (std::string)`  
  Get the structure containing regular cartesian grid data.

- `s_nebulosgrid getNebulousGridData() const throw (std::string)`  
  Get the structure containing nebulous grid data.

- `std::vector<s_monopolesource> getMonopoleSourceData() const`  
  Get a list of monopole source data structures.

- `std::vector<s_dipolesource> getDipoleSourceData() const`  
  Get a list of dipole source data structures.

- `std::vector<s_arbitarysource> getArbitrarySourceData() const`  
  Get a list of arbitrary source data structures.

- `std::vector<s_monopoledetector> getMonopoleDetectorData() const`  
  Get a list of monopole detector data structures.

- `std::vector<s_dipoledetector> getDipoleDetectorData() const`  
  Get a list of dipole detector data structures.

- `std::vector<s_arbitarydetector> getArbitraryDetectorData() const`  
  Get a list of arbitrary detector data structures.

- `std::vector<s_property> getPropertyData() const`  
  Get a list of property data structures.

- `BackgroundType getBackgroundType() const`  
  Get the type of background we are using.
- `std::string getConstantBGPropName() const`
  Get the name of the constant background property.

- `std::vector s_layer getBGLayers() const`
  Get a list of layered background data structures.

- `std::string getMapfilePath() const throw (std::string)`
  Get the node-property map file path.

- `std::string getArbitraryPropPath() const throw (std::string)`
  Get the arbitrary property assignment file path.

- `bool queryIsContiguouslyBound() const`
  Returns whether our boundary condition is applied contiguously.

- `s_boundary getContiguousBoundaryData() const throw (std::string)`
  Get the data structure for a contiguous boundary condition.

- `std::vector s_boundary getBoundaryData() const`
  Get a list of data structures for all boundaries (applied separately).

- `std::vector s_sphericalanomaly getSphericalAnomalyData() const`
  Get a list of spherical anomaly data structures.

- `std::vector s_cubicalanomaly getCubicalAnomalyData() const`
  Get a list of cubical anomaly data structures.

- `std::vector s_ellipsoidalanomaly getEllipsoidalAnomalyData() const`
  Get a list of ellipsoidal anomaly data structures.

- `std::vector s_circularanomaly getCircularAnomalyData() const`
  Get a list of circular anomaly data structures.

- `std::vector s_squareanomaly getSquareAnomalyData() const`
  Get a list of square anomaly data structures.

- `int getNormalPolyOrder() const`
  Get the order of polynomial used to fit normal vectors.

- `int getNormalNumNeighbors() const`
  Get the number of nearest neighbors used in normal vector estimation.
- double `getNormalEstTol` () const
  
  Get the tolerance for error in normal vector estimation.

- int `getSolveMaxIterations` () const
  
  Get the maximum number of iterations used in IML solving routines.

- double `getSolveResidualTolerance` () const
  
  Get the tolerance for residual in the solution.

- `s_precon` `getPreconditionerData` () const
  
  Get the data structure containing Preconditioner information.

- `FDDlib::enum Solver` `getSolverType` () const
  
  Get the type of IML solver we will use.

**Protected Methods**

- void `deblank` (char *str) const
  
  Remove whitespace from a string.

- `TagType` `getTagType` (char buf[]) const
  
  Get the tag type for the given string.

- `std::list<std::string>` `parseSeries` (std::string series, char del) const
  
  Parse a series contained within a string.

- void `parseGridTag` (int pos)
  
  Parse the grid tag, starting at the given file position.

- `FDDlib::gridSide3D` `parseGridSide` (char str[]) const
  
  Parse a string denoting a side of the grid.

- void `parseSourcesTag` (int pos)
  
  Parse the sources tag, starting at the given file position.

- void `parseMonopoleSource` (int pos)
  
  Parse a monopole source, starting at the given file position.

- void `parseDipoleSource` (int pos)
  
  Parse a dipole source, starting at the given file position.
- void parseArbitrarySource (int pos)
  
  Parse an arbitrary source, starting at the given file position.

- void parseDetectorsTag (int pos)
  
  Parse the detectors tag, starting at the given file position.

- void parseMonopoleDetector (int pos)
  
  Parse a monopole detector, starting at the given file position.

- void parseDipoleDetector (int pos)
  
  Parse a dipole detector, starting at the given file position.

- void parseArbitraryDetector (int pos)
  
  Parse an arbitrary detector, starting at the given file position.

- void parsePropertiesTag (int pos)
  
  Parse the properties tag, starting at the given file position.

- void parseMediumSetupTag (int pos)
  
  Parse the MediumSetup tag, starting at the given file position.

- void parseAnomaliesTag (int pos)
  
  Parse the anomalies tag, starting at the given file position.

- void parseBoundariesTag (int pos)
  
  Parse the boundaries tag, starting at the given file position.

- s_boundary parseBCType (char str[]) const
  
  Parse a boundary condition type.

- void parseOptionsTag (int pos)
  
  Parse the options tag, starting at the given file position.

- s_precon parsePreconditionerType (char str[]) const
  
  Parse a preconditioner type.

- FDDlib::enum_solver parseSolverType (char str[]) const
  
  Parse a solver type.

- void parseNodeListTag (int pos)
  
  Parse the nodeList tag, starting at the given position.
void parseNodeTag (int pos)
   Parse a single node tag, starting at the given position.

void parseDeltaListTag (std::ifstream &stream, int pos)
   Parse the deltalist tag in the given stream, at the given position.

void parseDeltaFile ()
   Parse the deltafile.

Protected Attributes

- char * filename
  The File in which to traverse through.

- std::ifstream file
  The file.

- Dimension domain
  dimensionality

- bool isContiguouslyBound
  boolean value telling us whether the boundary is a contiguous one

- std::vector< s_boundary > contiguousBoundary
  a single s_boundary if the boundary is contiguous

- std::vector< s_boundary > boundaries
  a vector of s_boundary’s if the boundaries are separately defined

- std::vector< s_sphericalanomaly > spherical_anomalies
  list of spherical anomaly data

- std::vector< s_cubicalanomaly > cubical_anomalies
  list of cubical anomaly data

- std::vector< s_ellipsoidalanomaly > ellipsoidal_anomalies
  list of ellipsoidal anomaly data

- std::vector< s_circularanomaly > circular_anomalies
  list of circular anomaly data
5.53 FDDparse::Parser Class Reference

- std::vector< s_squareanomaly > square_anomalies
  list of square anomaly data
- std::vector< s_node > nodes
  node data list
- std::vector< s_monopolesource > monopole_sources
  monopole source data list
- std::vector< s_dipolesource > dipole_sources
  dipole source data list
- std::vector< s_arbitrarysource > arbitrary_sources
  arbitrary source data list
- int currSrcId
  current source id
- std::vector< s_monopoledetector > monopole_detectors
  monopole detector data list
- std::vector< s_dipoledetector > dipole_detectors
  dipole detector data list
- std::vector< s_arbitrarydetector > arbitrary_detectors
  arbitrary detector data list
- int currDetId
  current detector id
- BackgroundType bg_type
  method we use to set up the background properties.
- char * mapfile_path
  mapping file if we have an mapped medium setup
- char * arbitrary_prop_path
  arbitrary property file if we’ve defined one
- std::vector< s_layer > layers
  s_layers instance if we have a layered background
- std::string constantprop_name
  name of property if we have a constant property

- int solver_iterations
  number of iterations to use in IML functions

- double solver_residualtol
  tolerance for solution residual

- s_precon solver_conditionertype
  matrix preconditioner type

- FDDlib::enum solver solver_type
  iterative solver type

- int n_normalOrder
  order of normal vector fit

- int n_numNeighbors
  number of boundary neighbors that will influence a normal vector fit

- double n_normalTol
  error tolerance for normal vector estimation

- std::vector< s_property > props
  list of property data

- GridType grid_type
  Grid type.

- s_arbcartgrid * acg
  ARG data if we have an arbitrary grid.

- s_regcartgrid * rcg
  RG data if we have a regular grid.

- s_nebulousgrid * ng
  nebulous grid data if we have a nebulous grid

- std::bitset< 10 > parsedTags
  bitstring to keep track of which tags we’ve parsed in
5.53 FDDparse::Parser Class Reference

5.53.1 Detailed Description

A class in which a configuration file can be defined and, once defined, traversed through.

Author:
Derek Uluski, Kyle Guilbert @change 4/11/03 Parser re-design to accommodate both 2d and 3d problems.

5.53.2 Constructor & Destructor Documentation

5.53.2.1 FDDparse::Parser::Parser (char * path, Dimension d) throw (std::string)

Constructor.

Parameters:
- *path* the path to the configuration file
- *d* dimensionality of the problem

5.53.3 Member Function Documentation

5.53.3.1 std::list< std::string > FDDparse::Parser::parseSeries (std::string series, char del) const [protected]

Parse a series contained within a string.

Parameters:
- *series* the series within a string
- *del* the delimiter

5.53.4 Member Data Documentation
5.53.4.1 **BackgroundType** FDDparse::Parser::bg_type  [protected]

method we use to set up the background properties.
this is better defined by which properties are assigned to the nodes before any anomalies are inserted.
The documentation for this class was generated from the following file:

- Parser.h
5.54 FDDparse::Parser::s_arbcartgrid Struct Reference

Stores all attributes of a semi-regular grid.

`#include <Parser.h>`

**Public Attributes**

- `int xdim`
  
  *X dimension.*

- `int ydim`
  
  *Y dimension.*

- `int zdim`
  
  *Z dimension.*

- `char * deltafile`
  
  *Stores the file name of a file for the deltalist.*

- `double espeed`
  
  *Nominal Speed of energy propagation.*

- `std::vector< double > deltax`
  
  *delta-x list*

- `std::vector< double > deltay`
  
  *delta-y list*

- `std::vector< double > deltaz`
  
  *delta-z list*

5.54.1 Detailed Description

Stores all attributes of a semi-regular grid.

The documentation for this struct was generated from the following file:
- Parser.h
An Arbitrary Detector Entry.

#include <Parser.h>

**Public Attributes**

- **int id**
  
  *detector id (used for ordering)*

- **std::vector< double > x**
  
  *Dynamic Array storing the X locations.*

- **std::vector< double > y**
  
  *Dynamic Array storing the Y locations.*

- **std::vector< double > z**
  
  *Dynamic Array storing the Z locations.*

- **std::vector< FDDlib::Complex< double > > weight**
  
  *Dynamic Array storing the complex weights.*

**5.55.1 Detailed Description**

An Arbitrary Detector Entry.

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

- Parser.h
5.56 FDDparse::Parser::s_arbitrarysource Struct Reference

An Arbitrary Source Entry.

#include <Parser.h>

Public Attributes

- int id
  source id (used for ordering)
- std::vector< double > x
  X location.
- std::vector< double > y
  Y location.
- std::vector< double > z
  Z location.
- std::vector< FDDlib::Complex< double > > val
  vector which stores all complex right hand side values
- std::vector< double > frequencies
  Dynamic Array which stores all frequencies.

5.56.1 Detailed Description

An Arbitrary Source Entry.

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

- Parser.h
boundary condition definition (mixedval may be unused)
#include <Parser.h>

Public Attributes

- **BoundType bt**
  
  *the boundary condition type*

- double **mixedval**
  
  *value for mixed boundary condition*

5.57.1 Detailed Description

boundary condition definition (mixedval may be unused)
The documentation for this struct was generated from the following file:

- Parser.h
5.58  FDDparse::Parser::s_circularanomaly Struct Reference

A Circular Anomaly Entry.
#include <Parser.h>

Public Attributes

- double xcenter
  X location.

- double ycenter
  Y location.

- double radius
  Radius of the spherical Anomaly.

- std::string prop
  String name of the property within the sphere.

5.58.1  Detailed Description

A Circular Anomaly Entry.

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

- Parser.h
A Cubical Anomaly Entry.

#include <Parser.h>

**Public Attributes**

- double `xcenter`
  
  *X location.*

- double `ycenter`
  
  *Y location.*

- double `zcenter`
  
  *Z location.*

- double `length`
  
  *Length of each side of the cube.*

- std::string `prop`
  
  *String name of the property within the cube.*

### 5.59.1 Detailed Description

A Cubical Anomaly Entry.

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

- Parser.h
5.60 FDDparse::Parser::s_dipoledetector Struct Reference

A Dipole Detector Entry.

#include <Parser.h>

Public Attributes

- int id
detector id (used for ordering)

- double x [2]
  X locations: index 0 for pole 1, index 1 for pole 2.

- double y [2]
  Y locations: index 0 for pole 1, index 1 for pole 2.

- double z [2]
  Z locations: index 0 for pole 1, index 1 for pole 2.

- FDDlib::Complex< double > weight [2]
  Dynamic Array storing the complex weights.

5.60.1 Detailed Description

A Dipole Detector Entry.
The documentation for this struct was generated from the following file:

- Parser.h
5.61 FDDparse::Parser::s_dipolesource Struct Reference

A dipole source entry.
#include <Parser.h>

Public Attributes

- int id
  
  \textit{source id (used for ordering)}

- double x [2]
  
  \textit{X locations: Index 0 for pole 1, index 1 for pole 2.}

- double y [2]
  
  \textit{Y locations: Index 0 for pole 1, index 1 for pole 2.}

- double z [2]
  
  \textit{Z locations: Index 0 for pole 1, index 1 for pole 2.}

- double magnitude [2]
  
  \textit{Magnitudes: Index 0 for pole 1, index 1 for pole 2.}

- double phase [2]
  
  \textit{Phases: Index 0 for pole 1, index 1 for pole 2.}

- std::vector< double > frequencies
  
  \textit{Dynamic Array that stores all frequency values.}

5.61.1 Detailed Description

A dipole source entry.

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

- Parser.h
5.62 FDDparse::Parser::s_ellipsoidalanomaly Struct Reference

An Ellipsoidal Anomaly Entry.
#include <Parser.h>

Public Attributes

- double xcenter
  X location.

- double ycenter
  Y location.

- double zcenter
  Z location.

- double xlen
  length of x semi-axis

- double ylen
  length of y semi-axis

- double zlen
  length of z semi-axis

- double theta1
  Euler angle 1.

- double theta2
  Euler angle 2.

- double theta3
  Euler angle 3.

- std::string prop
  String name of the property within the ellipsoid.
5.62.1 Detailed Description

An Ellipsoidal Anomaly Entry.

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

- Parser.h
5.63  FDDparse::Parser::s_layer Struct Reference

data we use to set up a layered background

#include <Parser.h>

Public Attributes

- int layerzdepth
  number of z-layers this layer spans

- std::string layerpropname
  name of property this layer uses

5.63.1  Detailed Description

data we use to set up a layered background

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

- Parser.h
A Monopole Detector Entry.

#include <Parser.h>

Public Attributes

- int id
  
  detector id (used for ordering)

- double xloc
  
  X location.

- double yloc
  
  Y location.

- double zloc
  
  Z location.

5.64.1 Detailed Description

A Monopole Detector Entry.

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

- Parser.h
5.65 FDDparse::Parser::s_monopolesource Struct Reference

A monopole source entry.
#include <Parser.h>

Public Attributes

- int id
  source id (used for ordering)
- double xloc
  X location.
- double yloc
  Y location.
- double zloc
  Z location.
- double magnitude
  Magnitude of the Monopole Source.
- double phase
  Phase of the Monopole Source.
- std::vector<double> frequencies
  Dynamic Array that stores all frequency values.

5.65.1 Detailed Description

A monopole source entry.
The documentation for this struct was generated from the following file:

- Parser.h
Stores all attributes of a nebulous grid.
#include <Parser.h>

**Public Attributes**

- int numnodes
  
  *Number of nodes contained in the arbitrary grid.*

- char * nodefile
  
  *The file name of a file for a nodelist.*

- double espeed
  
  *Floating-point value of energy speed within the grid.*

**5.66.1 Detailed Description**

Stores all attributes of a nebulous grid.

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

- Parser.h
Data structure to store all attributes of a node.

```cpp
#include <Parser.h>
```

### Public Attributes

- **int id**
  
  *node index*

- **double xloc**
  
  *X location.*

- **double yloc**
  
  *Y location.*

- **double zloc**
  
  *Z location.*

- **std::vector<int> neighbors**
  
  *vector that stores indeces of neighbors*

- **std::string property**
  
  *name of Property that the node will contain*

- **BoundType bcondition**
  
  *Boundary Condition that the node will contain.*

### Detailed Description

Data structure to store all attributes of a node.

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

- **Parser.h**
preconditioner data

#include <Parser.h>

Public Attributes

- PreconType type  
  type of preconditioner
- double relaxation  
  relaxation parameter for SSOR preconditioner

5.68.1 Detailed Description

preconditioner data

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

- Parser.h
5.69  **FDDparse::Parser::s_property Struct Reference**

A Property Entry.

```cpp
#include <Parser.h>
```

**Public Attributes**

- std::string `name`
  
  *Name of property.*

- bool `active`
  
  *Inactivity switch. (true=active, false=inactive).*

- std::string `coefs`
  
  *Property coefficients.*

### 5.69.1 Detailed Description

A Property Entry.

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

- `Parser.h`
5.70 FDDparse::Parser::s_regcartgrid Struct Reference

Stores all attributes of a regular grid.

#include <Parser.h>

Public Attributes

- int xdim
  - X dimension.

- int ydim
  - Y dimension.

- int zdim
  - Z dimension.

- double stepsize
  - Homogeneous grid spacing.

- double espeed
  - Nominal Speed of energy propagation.

5.70.1 Detailed Description

Stores all attributes of a regular grid.

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

- Parser.h
5.71 FDDparse::Parser::sphericalanomaly Struct Reference

A Spherical Anomaly Entry.
#include <Parser.h>

Public Attributes

- double xcenter
  \( X \) location.
- double ycenter
  \( Y \) location.
- double zcenter
  \( Z \) location.
- double radius
  Radius of the spherical Anomaly.
- std::string prop
  String name of the property within the sphere.

5.71.1 Detailed Description

A Spherical Anomaly Entry.
The documentation for this struct was generated from the following file:

- Parser.h
5.72 FDDparse::Parser::s_squareanomaly Struct Reference

A Square Anomaly Entry.
#include <Parser.h>

Public Attributes

- double xcenter
  X location.
- double ycenter
  Y location.
- double length
  Length of each side of the cube.
- std::string prop
  String name of the property within the cube.

5.72.1 Detailed Description

A Square Anomaly Entry.
The documentation for this struct was generated from the following file:

- Parser.h
5.73 FDDlib::Preconditioner Class Reference

Abstract base class for Preconditioners.

#include <Preconditioner.h>

Inheritance diagram for FDDlib::Preconditioner:

```
FDDlib::Preconditioner
```

```
FDDlib::DiagonalPreconditioner
FDDlib::ILU0Preconditioner
FDDlib::SSORPreconditioner
```

Public Methods

- **Preconditioner ()**  
  Constructor.

- **virtual ~Preconditioner ()**  
  Destructor.

- **virtual void init (SparseRowMatrix<double> &SRM)=0 throw (std::string)**  
  Initialize the preconditioner.

- **virtual RealVector<double> solve (const RealVector<double> &vec) const=0**  
  Solve the system.

Protected Attributes

- **int len_**
  Number of rows used.

5.73.1 Detailed Description

Abstract base class for Preconditioners.
Author:
Kyle Guilbert

## 5.73.2 Member Function Documentation

### 5.73.2.1 virtual void FDDlib::Preconditioner::init (SparseRowMatrix & SRM) throw (std::string) [pure virtual]

Initialize the preconditioner.

**Parameters:**
- **SRM** the Sparse Row Matrix

**Exceptions:**
- **string** if initialization fails

Implemented in FDDlib::DiagonalPreconditioner, FDDlib::ILU0Preconditioner, and FDDlib::SSORPreconditioner.

### 5.73.2.2 virtual RealVector<double> FDDlib::Preconditioner::solve (const RealVector<double> & vec) const [pure virtual]

Solve the system.

**Parameters:**
- **vec** Right hand side

Implemented in FDDlib::DiagonalPreconditioner, FDDlib::ILU0Preconditioner, and FDDlib::SSORPreconditioner.

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

- Preconditioner.h
5.74 FDDlib::Property Class Reference

Medium property class.

#include <Property.h>

Public Methods

- **Property ()**
  Default constructor.

- virtual ~Property ()
  Destructor.

- virtual double getSigma () const throw (std::string)
  Get the coefficient sigma. This is a real, double-precision number.

- virtual Complex< double > getKSquare (double c, double omega) const throw (std::string)
  Get the coefficient $k^2$.

- virtual void parseArgs (const char *args) throw (std::string)
  Parse any string arguments the property object might be passed from a file parser.

- bool isActive () const
  Determine whether this is an active Property.

- void setActive (bool active)
  Set the "active status" of this Property.

Protected Attributes

- bool active_
  Boolean value to denote whether this is an active property.

- double mu_a_
  Value of $\mu_a$ (absorption).
5.74 FDDlib::Property Class Reference

- double \texttt{\textit{g}}
  
  \textit{value of g}

- double \texttt{\textit{\mu_s}}
  
  \textit{value of \mu_s (reduced scattering)}.

5.74.1 Detailed Description

Medium property class.

Provides an interface for retrieving the mathematical coefficients used in the Helmholtz equation. This version of the \texttt{\textit{Property}} class is for Diffuse Optical Tomography (DOT). The format of the coefficient string parsed in parseArgs is: "absorbingCoeff gValue reducedScatteringCoeff"

\textbf{Author:}

Kyle Guilbert

5.74.2 Constructor & Destructor Documentation

5.74.2.1 FDDlib::Property::Property ()

Default constructor.

It may be useful to define more specific constructors if you are interfacing directly with the \texttt{\textit{Property}} class.

5.74.3 Member Function Documentation

5.74.3.1 virtual \texttt{\textit{Complex}}<\texttt{\textit{double}}> FDDlib::Property::getKSquare (\texttt{\textit{double}} \texttt{\textit{c}},

\texttt{\textit{double}} \texttt{\textit{omega}}) \texttt{\textit{const throw (std::string)}} \texttt{[virtual]}

Get the coefficient \(k^2\).

This is an imaginary, double-precision number. Since it is highly probable this conversion involves \texttt{\textit{c}} and \texttt{\textit{omega}}, we supply those as well.
Parameters:
   \(c\) Nominal speed of energy propagation through the Grid.
   \(\omega\) Frequency of source modulation

5.74.3.2 bool FDDlib::Property::isActive ()

Determine whether this is an active Property.

Returns:
   1 if this is an active property, 0 if it is not.

5.74.3.3 virtual void FDDlib::Property::parseArgs (const char * args) throw (std::string) [virtual]

Parse any string arguments the property object might be passed from a file parser.

Parameters:
   \(args\) the character-array of arguments

Exceptions:
   string if the argument is null

5.74.3.4 void FDDlib::Property::setActive (bool active)

Set the "active status" of this Property.

Parameters:
   active zero to make it inactive, non-zero to make it active.

5.74.4 Member Data Documentation
5.74.4.1 bool FDDlib::Property::active_ [protected]

Boolean value to denote whether this is an active property.
This helps to create irregularly shaped objects within the grid by enclosing them with inactive nodes.

Note:
This is true by default.

5.74.4.2 double FDDlib::Property::mu_a_ [protected]

double value of mu_a (absorption).
unit is 1/{spatial} where {spatial} is typically cm for DOT.

5.74.4.3 double FDDlib::Property::mu_s_ [protected]

double value of mu_s (reduced scattering).
unit is 1/{spatial} where {spatial} is typically cm for DOT.
The documentation for this class was generated from the following file:

• Property.h
5.75 FDDlib::RealVector< T > Class Template Reference

Class for storing and manipulating vectors of reals.

#include <RealVector.h>

Public Methods

- **RealVector** (int len)
  
  Constructor.

- **~RealVector** ()
  
  Destructor.

- **void deallocate ()**
  
  Deallocate this vector.

- **RealVector ()**
  
  Constructor.

- **void init (int len)**
  
  Initialize this vector to the given length.

- **RealVector (const RealVector &rv)**
  
  Copy constructor.

- **RealVector< T > & operator= (const RealVector< T > &rv)**
  
  Set this vector to the given vector.

- **RealVector< T > & operator= (const T v)**
  
  Set all elements of the vector to a given value.

- **RealVector< T > & operator+= (const RealVector< T > &rv)**
  
  Add a vector to this one. Assign the result to this vector.

- **RealVector< T > & operator-= (const RealVector< T > &rv)**
  
  Subtract a vector from this one. Assign the result to this vector.
RealVector< T >& operator *= (const T v)
Multiply this vector by another. Assign the result to this vector.

RealVector< T > operator+ (const RealVector< T >& rv) const
Add two real vectors, allocating space for a new one.

RealVector< T > operator- (const RealVector< T >& rv) const
Subtract two real vectors, allocating space for a new one.

RealVector< T > operator* (const T t) const
Multiply a vector by a scalar, allocating space for a new vector.

double dot (const RealVector< T >& rv) const
Compute the dot product of two vectors.

double norm () const
Compute the L2 norm of a vector.

double norm1 () const
Compute the L1 norm of a vector.

int size () const
Get the length (size) of this vector.

void set (int ind, T val)
Set a value in the vector.

T val (int ind) const
Get a value in the vector.

T & operator[] (int ind)
Overload the [] operator to get a value.

T operator[] (int ind) const
Overload the [] operator to get a value.

T max () const
Return the largest element of the vector.

T absmax () const
Return the largest element, in absolute value, of the vector.
T min () const

Return the smallest element of the vector.

Protected Attributes

- int len_
  
  length of the vector
- T * valVec_
  
  values

5.75.1 Detailed Description

template<class T> class FDDlib::RealVector<T>

Class for storing and manipulating vectors of reals.

Author:
  Greg Boverman 11-27-01 @change Added check for NULL vector in scalar =
  method 1-21-04 Jennifer Black @change Added check for memory allocation
  failure in RealVector(int) constructor @change Added check for memory allo-
  cation failure in init() @change Added check for memory allocation failure in
  RealVector(RealVector &) con.

5.75.2 Constructor & Destructor Documentation

5.75.2.1 template<class T> FDDlib::RealVector<T>::RealVector (int len)

Constructor.

Parameters:
  len  desired length of vector

5.75.3 Member Function Documentation
5.75.3.1  template<class T> T & FDDlib::RealVector< T >::operator() (int
ind)  [inline]

Overload the () operator to get a value.
This method is non-const: it returns a reference to the data.
The documentation for this class was generated from the following file:

- RealVector.h
5.76 FDDlib::RegularCartesianGrid2D Class Reference

An ArbitraryCartesianGrid2D with uniform distance between nodes.

```cpp
#include <RegularCartesianGrid2D.h>
```

Inheritance diagram for FDDlib::RegularCartesianGrid2D:

```
FDDlib::ArbitraryCartesianGrid2D
    ▼
     ▼
FDDlib::RegularCartesianGrid2D
```

Public Methods

- `RegularCartesianGrid2D(int numx, int numy, double energySpeed, double step) throw (std::string)`
  
  *Constructor.*

- `void setStep(double step) throw (std::string)`
  
  *Set the uniform step size.*

- `double getStep() const`
  
  *Get the step size.*

- `double averageDx() const`
  
  *Get the average Dx value.*

- `double averageDy() const`
  
  *Get the average Dy value.*

Protected Attributes

- `double step`
  
  *uniform step size*
5.76 FDDlib::RegularCartesianGrid2D Class Reference

5.76.1 Detailed Description

An ArbitraryCartesianGrid2D with uniform distance between nodes.

Author:
Kyle Guilbert

5.76.2 Constructor & Destructor Documentation

5.76.2.1 FDDlib::RegularCartesianGrid2D::RegularCartesianGrid2D (int numx, int numy, double energySpeed, double step) throw (std::string)

Constructor.

Parameters:
- numx  Number of nodes in the x direction. Range: [2, inf)
- numy  Number of nodes in the y direction. Range: [2, inf)
- energySpeed  Nominal speed of the radiation through this grid
- step  The uniform step size. Range: (0.0, inf)

5.76.3 Member Function Documentation

5.76.3.1 void FDDlib::RegularCartesianGrid2D::setStep (double step) throw (std::string)

Set the uniform step size.

Parameters:
- step  Double-precision uniform grid distance. Range: (0.0, inf)

Exceptions:
- string  if step is out of range

The documentation for this class was generated from the following file:
- RegularCartesianGrid2D.h
5.77 FDDlib::RegularCartesianGrid3D Class Reference

An ArbitraryCartesianGrid3D with uniform distance between nodes.

```cpp
#include <RegularCartesianGrid3D.h>
```

Inheritance diagram for FDDlib::RegularCartesianGrid3D:

```
FDDlib::ArbitraryCartesianGrid3D
FDDlib::RegularCartesianGrid3D
```

### Public Methods

- **RegularCartesianGrid3D** (int numx, int numy, int numz, double energySpeed, double step) throw (std::string)
  
  *Constructor.*

- void **setStep** (double step) throw (std::string)
  
  *Set the uniform step size.*

- double **getStep** () const
  
  *Get the step size.*

- double **averageDx** () const
  
  *Get the average Dx value.*

- double **averageDy** () const
  
  *Get the average Dy value.*

- double **averageDz** () const
  
  *Get the average Dz value.*
Protected Attributes

- double step_
  uniform step size

5.77.1 Detailed Description

An ArbitraryCartesianGrid3D with uniform distance between nodes.

Author:
Kyle Guilbert

5.77.2 Constructor & Destructor Documentation

5.77.2.1 FDDlib::RegularCartesianGrid3D::RegularCartesianGrid3D (int numx, int numy, int numz, double energySpeed, double step) throw (std::string)

Constructor.

Parameters:
- numx Number of nodes in the x direction. Range: [2, inf)
- numy Number of nodes in the y direction. Range: [2, inf)
- numz Number of nodes in the z direction. Range: [2, inf)
- energySpeed Nominal speed of the radiation through this grid
- step The uniform step size. Range: (0.0, inf)

5.77.3 Member Function Documentation

5.77.3.1 void FDDlib::RegularCartesianGrid3D::setStep (double step) throw (std::string)

Set the uniform step size.

Parameters:
- step Double-precision uniform grid distance. Range: (0.0, inf)
Exceptions:

- `string` if step is out of range

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

- `RegularCartesianGrid3D.h`
The abstract base source class.

#include <Source2D.h>

Inheritance diagram for FDDlib::Source2D:

```
FDDlib::Source2D
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
FDDlib::ArbitrarySource2D FDDlib::DipoleSource2D FDDlib::MonopoleSource2D
```

Public Methods

- **Source2D ()**
  *Default constructor.*

- **Source2D (const std::vector<double> &frequencies)**
  *Constructor.*

- **virtual ~Source2D ()**
  *Destructor.*

- **void setFrequencies (const std::vector<double> &frequencies)**
  *Set the list of frequencies.*

- **int getNumFrequencies () const**
  *Get the number of frequencies this Source2D operates at.*

- **double getFrequency (int num) const throw (std::string)**
  *Get a frequency from the list.*

- **std::vector<double> getFrequencies () const**
  *Get the entire list of frequencies.*

- **bool canOperateAt (double frequency) const**
  *Returns whether this source can operate at the given frequency.*
virtual Complex< double > getRightHandSide (const CartesianNode2D &node) const = 0

Get the complex right-hand-side associated with the given node.

Protected Attributes

- std::vector< double > freq_

The list of frequencies this source operates at.

5.78.1 Detailed Description

The abstract base source class.

Here we declare an interface by which the toolkit can extract the complex right hand side of the PDE equation, given that the source implementation is concerned with the CartesianNode2D at which the equation is focused.

Author:
Kyle Guilbert

5.78.2 Constructor & Destructor Documentation

5.78.2.1 FDDlib::Source2D::Source2D (const std::vector< double > & frequencies)

Constructor.

Parameters:
 frequencies List of frequency modulations.

5.78.3 Member Function Documentation

5.78.3.1 double FDDlib::Source2D::getFrequency (int num) const throw
(std::string)

Get a frequency from the list.
Parameters:
- \textit{num} Index of frequency in the list. Range: \([0, \text{getNumFrequencies}())\) @bugfix 12/08/02 Rearranged this function for better error checking.

Exceptions:
- \textit{string} if argument \textit{num} is out of range

5.78.3.2 virtual \textbf{Complex}<$\text{double}$> FDDlib::Source2D::getRightHandSide
(const \textbf{CartesianNode2D} & \textit{node}) const [pure virtual]

Get the complex right-hand-side associated with the given node.

Parameters:
- \textit{node} The node (in 2D space) of focus. We formulate the right hand side based on whether the node is part of our distribution of locations.

Implemented in FDDlib::ArbitrarySource2D, FDDlib::DipoleSource2D, and FDDlib::MonopoleSource2D.

5.78.3.3 void FDDlib::Source2D::setFrequencies (const std::vector<$\text{double}$> & \textit{frequencies})

Set the list of frequencies.

Parameters:
- \textit{frequencies} Frequency list

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

- Source2D.h
5.79 FDDlib::Source3D Class Reference

The abstract base source class.

```cpp
#include <Source3D.h>
```

Inheritance diagram for FDDlib::Source3D:

```
FDDlib::Source3D
     |        |
     FDDlib::ArbitrarySource3D FDDlib::DipoleSource3D FDDlib::MonopoleSource3D
```

**Public Methods**

- **Source3D ()**  
  *Default constructor.*

- **Source3D (const std::vector<double> &frequencies)**  
  *Constructor.*

- **virtual ~Source3D ()**  
  *Destructor.*

- **void setFrequencies (const std::vector<double> &frequencies)**  
  *Set the list of frequencies.*

- **int getNumFrequencies () const**  
  *Get the number of frequencies this Source3D operates at.*

- **double getFrequency (int num) const throw (std::string)**  
  *Get a frequency from the list.*

- **std::vector<double> getFrequencies () const**  
  *Get the entire list of frequencies.*

- **bool canOperateAt (double frequency) const**  
  *Returns whether this source can operate at the given frequency.*
virtual Complex< double > getRightHandSide (const CartesianNode3D &node) const = 0

Get the complex right-hand-side value associated with the given node.

Protected Attributes

std::vector< double > freq_

The list of frequencies this source operates at.

5.79.1 Detailed Description

The abstract base source class.

Here we declare an interface by which the toolkit can extract the complex right hand side of the PDE equation, given that the source implementation is concerned with the CartesianNode3D at which the equation is focused.

Author:
Kyle Guilbert

5.79.2 Constructor & Destructor Documentation

5.79.2.1 FDDlib::Source3D::Source3D (const std::vector< double > & frequencies)

Constructor.

Parameters:
 frequencies list of frequency modulations.

5.79.3 Member Function Documentation

5.79.3.1 double FDDlib::Source3D::getFrequency (int num) const throw (std::string)

Get a frequency from the list.
Parameters:
   num   Index of frequency in the list. Range: \([0, \text{getNumFrequencies}())\)  
12/08/02 Rearranged this function for better error checking.

Exceptions:
   string  if argument num is out of range

\textbf{5.79.3.2} virtual \texttt{Complex\textangle double \rangle} \texttt{FDDlib::Source3D::getRightHandSide} \texttt{(const CartesianNode3D & node) const}  
[pure virtual]

Get the complex right-hand-side value associated with the given node.

Parameters:
   node  the node (in 3D space) of focus. We formulate the right hand side based on 
whether the node is part of our distribution of locations.

Implemented in \texttt{FDDlib::ArbitrarySource3D}, \texttt{FDDlib::DipoleSource3D}, and 
\texttt{FDDlib::MonopoleSource3D}.

\textbf{5.79.3.3} void \texttt{FDDlib::Source3D::setFrequencies (const std::vector\textangle double \rangle} \texttt{& frequencies)}

Set the list of frequencies.

Parameters:
   frequencies  Frequency list

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

- Source3D.h
5.80  
FDDlib::SparseRowEntry< T > Struct Template Reference

A single entry in the SparseRowMatrix.
#include <SparseRowMatrix.h>

Public Attributes

- int column
  which column in the row

- T val
  entry value

5.80.1  Detailed Description

template<class T> struct FDDlib::SparseRowEntry< T >

A single entry in the SparseRowMatrix.
Since these are arranged in an array, it is useful to access an entry externally and traverse through memory to access others.

Author:
Greg Boverman

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

- SparseRowMatrix.h
5.81 FDDlib::SparseRowMatrix< T > Class Template Reference

A sparse matrix, with elements stored in row-column form.

#include <SparseRowMatrix.h>

Inheritance diagram for FDDlib::SparseRowMatrix< T >:

```
FDDlib::Matrix< T >

FDDlib::SparseRowMatrix< T >

FDDlib::DynamicSparseRowMatrix< T >
```

Public Methods

- **SparseRowMatrix ()**
  
  constructor

- **SparseRowMatrix (int maxrows, int max_per_row)**
  
  constructor

- **SparseRowMatrix (const SparseRowMatrix< T >&SRM)**
  
  constructor

- **virtual ~SparseRowMatrix ()**
  
  destructor

- **virtual SparseRowMatrix< T >& operator=(const SparseRowMatrix< T >&SRM)**
  
  = operator overloader

- **virtual void init (int maxrows, int max_per_row)**
  
  initialize the matrix

- **virtual void deallocate ()**
Deallocate memory for the matrix.

- virtual T val (int row, int col) const
  Return a value for the specified row and column.

- virtual int getEntriesPerRow () const
  Get the number of entries per row.

- virtual SparseRowEntry< T >*getRowEntry (int row) const
  Get a pointer to the first entry of a row.

- virtual int getCol (int row, int n) const
  Get the index of the n’th column in the sparse matrix.

- virtual int getCol (int row, int n, T &v) const
  Get the index of the n’th column in the sparse matrix Also, get the value at this index.

- virtual int getNumCols (int row) const
  Get the number of non-zero columns for a row.

- virtual int getNumCols () const
  Get the max number of columns in the matrix.

- virtual int getNumRows () const
  Get the number of rows in the matrix.

- virtual void set (int row, int col, T val) throw (std::string)
  Set a row and column.

- template<class Vector> Vector operator * (const Vector &m) const
  Matrix-vector multiply.

- RealVector< T >getRow (int row) const
  get an entire row of the matrix

- RealVector< T >getColumn (int col) const
  get an entire column of the matrix
Protected Attributes

- int MaxRows_
  rows in the matrix
- int EntriesPerRow_
  how many entries we can actually store
- int * NumCols_
  array of column counts for every row
- SparseRowEntry< T > * Entries_
  the array of entries

5.81.1 Detailed Description

template<class T> class FDDlib::SparseRowMatrix< T >

A sparse matrix, with elements stored in row-column form.

A class for a dynamically allocated sparse matrix class in row format, meaning that for
each row we have a pointer to vectors of column indices and values. For efficiency,
we have a member to add an element without checking whether that element is already
defined (the burden is on the user to make sure that the library is being used correctly).

5.81.2 Member Function Documentation

5.81.2.1 template<class T> template<class Vector> Vector
FDDlib::SparseRowMatrix< T >::operator * (const Vector & m) const
[inline]

Matrix-vector multiply.
Code is fairly low-level for maximal efficiency.

5.81.2.2 template<class T> void FDDlib::SparseRowMatrix< T >::set (int
row, int col, T val) throw (std::string) [inline, virtual]

Set a row and column.
@bugfix 12/09/02 (Kyle Guilbert) check for row,col >= 0
Implements FDDlib::Matrix< T >.
Reimplemented in FDDlib::DynamicSparseRowMatrix< T >.

5.81.2.3 template<class T> T FDDlib::SparseRowMatrix< T >::val (int row, int col) const [inline, virtual]

Return a value for the specified row and column.
If no value is defined, return 0.
Implements FDDlib::Matrix< T >.
The documentation for this class was generated from the following file:

- SparseRowMatrix.h
5.82 FDDlib::SphericalAnomaly Class Reference

Acts as a spherical inhomogeneity in the medium.

#include <SphericalAnomaly.h>

Inheritance diagram for FDDlib::SphericalAnomaly:

```
FDDlib::Anomaly3D
```

```
FDDlib::SphericalAnomaly
```

Public Methods

- **SphericalAnomaly** (const Location3D &center, double radius, Property *prop) throw (std::string)
  
  Constructor.

- **Location3D getCenter** () const
  
  Get the location of this spherical anomaly.

- void **setCenter** (const Location3D &center)
  
  Set the location of this spherical anomaly.

- double **getRadius** () const
  
  Get the radius of this spherical anomaly.

- void **setRadius** (double radius) throw (std::string)
  
  Set the radius of this spherical anomaly.

- bool **encloses** (const CartesianNode3D &node) const
  
  Determine whether this sphere encloses the given node.

Protected Attributes

- **Location3D center**
the location (in 3D space) of the sphere’s center

- double radius_,
  radius of sphere

5.82.1 Detailed Description

Acts as a spherical inhomogeneity in the medium.

Author:
Kyle Guilbert 7/30/02, Derek Uluski @change 12/04/02 Added a bunch of set/get functions.

5.82.2 Constructor & Destructor Documentation

5.82.2.1 FDDlib::SphericalAnomaly::SphericalAnomaly (const Location3D & center, double radius, Property * prop) throw (std::string)

Constructor.

Parameters:
- center 3d location of the sphere’s center
- radius radius of sphere (range: positive values)
- prop pointer to Property object associated with the sphere

5.82.3 Member Function Documentation

5.82.3.1 bool FDDlib::SphericalAnomaly::encloses (const CartesianNode3D & node) const [virtual]

Determine whether this sphere encloses the given node.

Parameters:
- node the CartesianNode3D

Return values:
- true if this sphere encloses the node
false if this sphere does not enclose the node

Implements FDDlib::Anomaly3D.

5.82.3.2 void FDDlib::SphericalAnomaly::setCenter (const Location3D & center)

Set the location of this spherical anomaly.

Parameters:
- xcenter x coordinate of center
- ycenter y coordinate of center
- zcenter z coordinate of center

5.82.3.3 void FDDlib::SphericalAnomaly::setRadius (double radius) throw (std::string)

Set the radius of this spherical anomaly.

Parameters:
- radius the radius. Range: [0.0, inf)

Exceptions:
- string if radius is out of range

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

- SphericalAnomaly.h
5.83 FDDlib::SquareAnomaly Class Reference

Acts as a cubical inhomogeneity in the medium.

#include <SquareAnomaly.h>

Inheritance diagram for FDDlib::SquareAnomaly:

```
FDDlib::Anomaly2D
  |
  v
FDDlib::SquareAnomaly
```

Public Methods

- **SquareAnomaly** (const Location2D &center, double length, Property *prop) throw (std::string)
  Constructor.

- void **setCenter** (const Location2D &center)
  Set the location of this square anomaly.

- **Location2D getCenter** () const
  Get the location of this square anomaly.

- double **getLength** () const
  Get the length of this cubical anomaly.

- void **setLength** (double length) throw (std::string)
  Set the length of this square anomaly.

- bool **encloses** (const CartesianNode2D &node) const
  Determine whether this square encloses the given node.

Protected Attributes

- **Location2D center_**
the location (in 2D space) of the square’s center

- double \( \text{len}_x \)
  
  length of each side

5.83.1 Detailed Description

Acts as a cubical inhomogeneity in the medium.

Author:
Kyle Guilbert 7/30/02 , Derek Uluski @change 12/04/02 Added lots of set/get functions

5.83.2 Constructor & Destructor Documentation

5.83.2.1 FDDLib::SquareAnomaly::SquareAnomaly (const Location2D & center, double length, Property * prop) throw (std::string)

Constructor.

Parameters:
- center 2d location of the square’s center
- length length of sides. Range: [0.0, inf)
- prop pointer to the Property object

5.83.3 Member Function Documentation

5.83.3.1 bool FDDLib::SquareAnomaly::encloses (const CartesianNode2D & node) const [virtual]

Determine whether this square encloses the given node.

Parameters:
- \( p \) the CartesianNode2D
5.83 FDDlib::SquareAnomaly Class Reference

Return values:

- **true** if this cube encloses the CartesianNode2D
- **false** if this cube does not enclose the CartesianNode2D

Implements FDDlib::Anomaly2D.

5.83.3.2 void FDDlib::SquareAnomaly::setCenter (const Location2D & center)

Set the location of this square anomaly.

**Parameters:**

- **center** 2d location of the square’s center

5.83.3.3 void FDDlib::SquareAnomaly::setLength (double length) throw (std::string)

Set the length of this square anomaly.

**Parameters:**

- **length** the length of the sides of the square. Range: [0.0, inf)

**Exceptions:**

- **string** if length is out of range

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

- SquareAnomaly.h
5.84  FDDlib::SSORPreconditioner Class Reference

Implement an SSOR preconditioner.

#include <SSORPreconditioner.h>

Inheritance diagram for FDDlib::SSORPreconditioner::

```
FDDlib::Preconditioner
\downarrow
FDDlib::SSORPreconditioner
```

Public Methods

- **SSORPreconditioner** (double omega)
  
  constructor

- **~SSORPreconditioner** ()
  
  destructor

- void **init** (SparseRowMatrix< double > &SRMD) throw (std::string)
  
  initialize the preconditioner data

- template<class Vector> Vector **lowerSolve** (const Vector &x) const
  
  Solve the lower-triangular system: 
  \( (D - \omega E)^{-1}y = x \).

- template<class Vector> Vector **diagonalMultiply** (const Vector &x) const
  
  Multiply a vector by the diagonal of the original matrix.

- **RealVector< double > upperSolve** (const RealVector< double > &x) const
  
  Solve the upper-triangular system: 
  \( (D - \omega F)^{-1}y = x \).

- **RealVector< double > solve** (const RealVector< double > &x) const
  
  Solve the system by doing a lower solve followed by an upper solve.
5.84 FDDlib::SSORPreconditioner Class Reference

5.84.1 Detailed Description

Implement an SSOR preconditioner.

Essentially, this preconditioner solves: $M^{1} = (D - \omega F)^{1} D (D - \omega E)^{-1}$

Where E is the lower diagonal part of the original matrix and F is the upper diagonal part.

Author:
Greg Boverman 10-19-01

5.84.2 Member Function Documentation

5.84.2.1 template<class Vector> Vector
FDDlib::SSORPreconditioner::diagonalMultiply
(const Vector & x) const [inline]

Multiply a vector by the diagonal of the original matrix.

Parameters:
  x right hand side

Return values:
  x *invDiag

5.84.2.2 void FDDlib::SSORPreconditioner::init (SparseRowMatrix< double > & SRMD) throw (std::string) [inline, virtual]

initialize the preconditioner data

Parameters:
  SRMD our extensible sparse row matrix

Implements FDDlib::Preconditioner.

5.84.2.3 template<class Vector> Vector FDDlib::SSORPreconditioner::lowerSolve (const Vector & x) const [inline]

Solve the lower-triangular system: $(D - \omega E)^{-1}y = x.$
Parameters:
   \( x \) right hand side

Return values:
   \( the \) solution vector

5.84.2.4 \texttt{RealVector\langle double \rangle FDDlib::SSORPreconditioner::solve (const RealVector\langle double \rangle & x) const [inline, virtual]}

Solve the system by doing a lower solve followed by an upper solve.

Parameters:
   \( x \) right hand side

Return values:
   \( the \) solution vector

Implements \texttt{FDDlib::Preconditioner}.

5.84.2.5 \texttt{RealVector\langle double \rangle FDDlib::SSORPreconditioner::upperSolve (const RealVector\langle double \rangle & x) const [inline]}

Solve the upper-triangular system: \((D - \omega F)^{-1} y = x\).

Parameters:
   \( x \) right hand side

Return values:
   \( the \) solution vector

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

- \texttt{SSORPreconditioner.h}
## Index

<table>
<thead>
<tr>
<th>Anomaly2D</th>
<th>FDDlib::Anomaly2D, 21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anomaly3D</td>
<td>FDDlib::Anomaly3D, 24</td>
</tr>
<tr>
<td>ArbitraryCartesianGrid2D</td>
<td>FDDlib::ArbitraryCartesianGrid2D, 27</td>
</tr>
<tr>
<td>ArbitraryCartesianGrid3D</td>
<td>FDDlib::ArbitraryCartesianGrid3D, 36</td>
</tr>
<tr>
<td>BoundaryCondition2D</td>
<td>FDDlib::BoundaryCondition2D, 75</td>
</tr>
<tr>
<td>BoundaryCondition3D</td>
<td>FDDlib::BoundaryCondition3D, 78</td>
</tr>
<tr>
<td>Complex</td>
<td>FDDlib::Complex, 94</td>
</tr>
<tr>
<td>ComplexVector</td>
<td>FDDlib::ComplexVector, 101</td>
</tr>
<tr>
<td>DenseMatrix</td>
<td>FDDlib::DenseMatrix, 110</td>
</tr>
<tr>
<td>Detector2D</td>
<td>FDDlib::Detector2D, 114</td>
</tr>
<tr>
<td>Detector3D</td>
<td>FDDlib::Detector3D, 116</td>
</tr>
<tr>
<td>DiagonalPreconditioner</td>
<td>FDDlib::DiagonalPreconditioner, 118</td>
</tr>
<tr>
<td>ILU0Preconditioner</td>
<td>FDDlib::ILU0Preconditioner, 170</td>
</tr>
<tr>
<td>Preconditioner</td>
<td>FDDlib::Preconditioner, 226</td>
</tr>
<tr>
<td>Property</td>
<td>FDDlib::Property, 228</td>
</tr>
<tr>
<td>RealVector</td>
<td>FDDlib::RealVector, 232</td>
</tr>
<tr>
<td>SSORPreconditioner</td>
<td>FDDlib::SSORPreconditioner, 258</td>
</tr>
<tr>
<td>Source2D</td>
<td>FDDlib::Source2D, 241</td>
</tr>
<tr>
<td>Source3D</td>
<td>FDDlib::Source3D, 244</td>
</tr>
<tr>
<td>SparseRowMatrix</td>
<td>FDDlib::SparseRowMatrix, 248</td>
</tr>
<tr>
<td>a-</td>
<td>FDDlib::BCMixed2D, 66</td>
</tr>
<tr>
<td>acg</td>
<td>FDDlib::BCMixed3D, 70</td>
</tr>
<tr>
<td>achievedToL</td>
<td>FDDlib::FiniteDifferences2D, 150</td>
</tr>
<tr>
<td>active</td>
<td>FDDlib:: FiniteDifferences3D, 161</td>
</tr>
<tr>
<td>active</td>
<td>FDDlib:: Property, 230</td>
</tr>
<tr>
<td>activeAndInternal</td>
<td>FDDlib::BCMixed2D, 66</td>
</tr>
<tr>
<td>activeNeighbor</td>
<td>FDDlib:: BCMixed3D, 70</td>
</tr>
<tr>
<td>addLocation</td>
<td>FDDlib:: CartesianNode2D, 83</td>
</tr>
<tr>
<td>activeNeighbor</td>
<td>FDDlib:: CartesianNode3D, 88</td>
</tr>
</tbody>
</table>
INDEX

FDDlib::ArbitraryDetector2D, 46
FDDlib::ArbitraryDetector3D, 50
FDDlib::ArbitrarySource2D, 54
FDDlib::ArbitrarySource3D, 58
FDDparse::Parser, 201
arbitrary_prop_path
arbitrary_sources
FDDlib::ArbitraryCartesianGrid2D
FDDlib::ArbitraryCartesianGrid3D
arbitrary
ArbitraryDetector2D
ArbitrarySource2D
FDDlib::ArbitraryDetector2D, 45
ArbitraryDetector3D
ArbitrarySource3D
FDDlib::ArbitraryDetector3D, 49
FDDlib::ArbitrarySource2D, 53
FDDlib::ArbitrarySource3D, 54
FDDlib::ArbitrarySource3D, 57
FDDlib::ArbitrarySource3D, 58
assignContiguously
FDDlib::BoundaryCondition2D, 76
FDDlib::BoundaryCondition3D, 79
assignToNode
FDDlib::BoundaryCondition2D, 76
FDDlib::BoundaryCondition3D, 79
assignToSide
FDDlib::BoundaryCondition2D, 77
FDDlib::BoundaryCondition3D, 80
averageDx
FDDlib::BoundaryCondition2D, 61
FDDlib::BoundaryCondition3D, 63
FDDlib::BoundaryCondition3D, 67
FDDlib::BoundaryCondition3D, 71
FDDlib::BoundaryCondition2D, 76
FDDlib::BoundaryCondition3D, 79
arbitrary detectors
FDDlib::FiniteDifferences2D, 150
FDDlib::FiniteDifferences3D, 161
FDDlib::FiniteDifferences2D, 151
FDDlib::FiniteDifferences3D, 162
FDDlib::FiniteDifferences2D, 150
FDDlib::FiniteDifferences3D, 161
analyticalNormals
FDDlib::ArbitraryCartesianGrid2D, 31
FDDlib::ArbitraryCartesianGrid3D, 40
assignContiguously
FDDlib::BCDirichlet2D, 61
FDDlib::BCDirichlet3D, 63
assignToNode
FDDlib::BCDirichlet2D, 61
FDDlib::BCDirichlet3D, 63
assignToSide
FDDlib::BCMixed2D, 67
FDDlib::BCMixed3D, 71
averageDx
FDDlib::BCMixed2D, 67
FDDlib::BCMixed3D, 71
arbitrary detectors
FDDlib::ComplexVector, 102
constantprop
FDDparse::Parser, 202
contiguousBoundary
FDDparse::Parser, 200
copySubMatrices
FDDlib::FiniteDifferences2D, 149
FDDlib::FiniteDifferences3D, 160
createLayeredBackground
FDDlib::ArbitraryCartesianGrid3D, 41
cubical anomalies
FDDparse::Parser, 200
CubicalAnomaly
FDDlib::CubicalAnomaly, 108
currDetId
FDDparse::Parser, 201
currFreq
FDDlib::FiniteDifferences2D, 150
FDDlib::FiniteDifferences3D, 161
currSrcId
FDDparse::Parser, 201
currSrcNum
FDDlib::FiniteDifferences2D, 150
FDDlib::FiniteDifferences3D, 161
D
FDDlib::EllipsoidalAnomaly, 145
data
FDDlib::ArbitraryDetector2D::s_adentry, 48
FDDlib::ArbitraryDetector3D::s_adentry, 52
data1
FDDlib::DipoleDetector2D, 121
FDDlib::DipoleDetector3D, 126
data2
FDDlib::MonopoleDetector2D, 182
FDDlib::MonopoleDetector3D, 185
deallocate
FDDlib::ComplexVector, 101
FDDlib::DenseMatrix, 110
FDDlib::Matrix, 179
FDDlib::RealVector, 232
FDDlib::SparseRowMatrix, 248
deblank
FDDparse::Parser, 198
DEFAULT_SPACING
FDDlib::ArbitraryCartesianGrid2D, 29
FDDlib::ArbitraryCartesianGrid3D, 38
deltafile
FDDparse::Parser::arbcartgrid, 205
deltax
FDDlib::ArbitraryCartesianGrid2D, 30
FDDlib::ArbitraryCartesianGrid3D, 39
FDDparse::Parser::arbcartgrid, 205
deltay
FDDlib::ArbitraryCartesianGrid2D, 30
FDDlib::ArbitraryCartesianGrid3D, 39
FDDparse::Parser::arbcartgrid, 205
deltaz
FDDlib::ArbitraryCartesianGrid3D,
property_ 22
setProperty, 23
FDDlib::Anomaly3D, 24
~Anomaly3D, 24
Anomaly3D, 25
enclorses, 25
getProperty, 24
setProperty, 25
FDDlib::ArbitraryCartesianGrid2D, 27
~ArbitraryCartesianGrid2D, 27
averageDx, 27
averageDy, 28
DEFAULT_SPACING, 29
deltax, 30
deltay, 30
energySpeed_, 29
getAllBorders, 28
getDxList, 27
getDyList, 27
getEnergySpeed, 28
getNode, 28
nodeList_, 29
numActiveNodes, 29
numActiveNodes_, 29
numBorderNodes, 29
numNodes, 28
numX, 28
numx_, 29
numY, 28
numy_, 29
setEnergySpeed, 28
setupNodes, 29
UNKNOWN_NUMACTIVE, 30
FDDlib::ArbitraryCartesianGrid2D
analyticalNormals, 31
ArbitraryCartesianGrid2D, 30
computeNormals, 31
fillUniformProperty, 32
generateLinearIndices, 32
getBorderBySide, 32
insertAnomaly, 32
offsetsToIndex, 33
setDelta, 33
FDDlib::ArbitraryCartesianGrid3D::s_::offsets, 35
xoffset, 35
yoffset, 35
FDDlib::ArbitraryCartesianGrid3D, 36
~ArbitraryCartesianGrid3D, 36
averageDx, 37
averageDy, 37
averageDz, 37
DEFAULT_SPACING, 38
deltax, 39
deltay, 39
deltaz, 39
energySpeed_, 39
generateLinearIndices, 38
getAllBorders, 37
getDxList, 36
getDyList, 36
getDzList, 36
getEnergySpeed, 38
getNode, 37
nodeList_, 39
numActiveNodes, 38
numActiveNodes_, 39
numBorderNodes, 38
numNodes, 37
numX, 37
numx_, 39
numY, 37
numy_, 39
numZ, 37
numz_, 39
setEnergySpeed, 38
setupNodes, 38
UNKNOWN_NUMACTIVE, 39
FDDlib::ArbitraryCartesianGrid3D
analyticalNormals, 40
ArbitraryCartesianGrid3D, 40
computeNormals, 41
createLayeredBackground, 41
fillUniformProperty, 42
generateLinearIndex, 42
setDelta, 43
FDDlib::ArbitraryCartesianGrid3D::s_::offsets, 44
xoffset, 44
yoffset, 44
zoffset, 44
FDDlib::ArbitraryDetector2D, 45
aperture, 46
ArbitraryDetector2D, 45
getApertureSize, 45
FDDlib::ArbitraryDetector2D::addLocation, 46
data, 46
getEntry, 47
setData, 47
FDDlib::ArbitraryDetector2D::s_::adentry, 48
data, 48
location, 48
weight, 48
FDDlib::ArbitraryDetector3D, 49
aperture, 50
ArbitraryDetector3D, 49
getApertureSize, 49
FDDlib::ArbitraryDetector3D::addLocation, 50
data, 50
getEntry, 51
setData, 51
FDDlib::ArbitraryDetector3D::s_::adentry, 52
data, 52
location, 52
weight, 52
FDDlib::ArbitrarySource2D, 53
ArbitrarySource2D, 53
distribution, 54
getFieldSize, 53
FDDlib::ArbitrarySource2D::addLocation, 54
ArbitrarySource2D, 54
data, 54
getEntry, 54
getRightHandSide, 55
FDDlib::ArbitrarySource2D::s_::asentry, 56
data, 56
location, 56
value, 56
FDDlib::ArbitrarySource3D, 57
ArbitrarySource3D, 57
distribution, 58
getFieldSize, 57
FDDlib::ArbitrarySource3D::addLocation, 58
ArbitrarySource3D, 58
data, 58
getRightHandSide, 59
FDDlib::ArbitrarySource3D::s_::asentry, 60
data, 60
location, 60
value, 60
FDDlib::BCDirichlet2D, 61
apply, 61
FDDlib::BCDirichlet3D, 63
apply, 63
FDDlib:: BCMixed2D, 65
a_., 66
activeAndInternal, 66
apply, 67
BCMixed2D, 66
data, 65
resetData, 65
setA, 65
UNKNOWN_VALUE, 66
FDDlib:: BCMixed2D::s_::asentry, 68
n_borderNode, 68
n_linearIndex, 68
n_node, 68
FDDlib:: BCMixed3D, 69
a_., 70
activeAndInternal, 70
apply, 71
BCMixed3D, 70
data, 69
resetDataVector, 69
setA, 69
UNKNOWN_VALUE, 70
FDDlib:: BCMixed3D::s_::asentry, 72
n_borderNode, 72
n_linearIndex, 72
n_node, 72
FDDlib::BCNeumann2D, 73

Generated on Mon Aug 30 15:41:16 2004 for FDDLib by Doxygen
BCNeumann2D, 73
FDDlib::BCNeumann3D, 74
BCNeumann3D, 74
FDDlib::BoundaryCondition2D, 75
~BoundaryCondition2D, 75
nodes, 76
FDDlib::BoundaryCondition2D
apply, 76
assignContiguously, 76
assignToNode, 76
assignToSide, 77
FDDlib::BoundaryCondition3D, 78
~BoundaryCondition3D, 78
nodes, 79
FDDlib::BoundaryCondition3D
apply, 79
assignContiguously, 79
assignToNode, 79
assignToSide, 80
FDDlib::CartesianNode2D, 81
getLocation, 81
getNormalVector, 81
getNumNeighbors, 81
getProperty, 81
INACTIVE_INDEX, 82
isBorderNode, 82
linearIndex, 82
location, 82
neighbors, 82
normalVector, 82
property, 82
UNKNOWN_INDEX, 83
FDDlib::CartesianNode2D
activeNeighbor, 83
CartesianNode2D, 83
calculateLocation, 83
distanceTo, 84
getLinearIndex, 84
getNeighbor, 84
setLinearIndex, 84
setNeighbor, 85
setNormalVector, 85
setProperty, 85
FDDlib::CartesianNode3D, 86
getLocation, 86
getNormalVector, 86
FDDlib::CircularAnomaly, 91
center, 91
calculateCenter, 91
calculateRadius, 91
FDDlib::CircularAnomaly
CircularAnomaly, 92
encloses, 92
setCenter, 93
setRadius, 93
FDDlib::Complex, 94
~Complex, 94
Complex, 94, 96
conjugate, 96
calculateImag, 94
calculateReal, 94
imagPart, 96
magnitude, 95
operator *, 97
operator ==, 97
operator+, 98
operator+=, 98
operator-, 98
operator-=, 98
operator/, 99
<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
<th>Class/Package</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator/=</td>
<td>99</td>
<td>operator/=</td>
<td></td>
</tr>
<tr>
<td>operator=</td>
<td>100</td>
<td>operator=</td>
<td></td>
</tr>
<tr>
<td>realPart</td>
<td>96</td>
<td>realPart</td>
<td></td>
</tr>
<tr>
<td>setImag</td>
<td>94</td>
<td>setImag</td>
<td></td>
</tr>
<tr>
<td>setReal</td>
<td>94</td>
<td>setReal</td>
<td></td>
</tr>
<tr>
<td>squareMagnitude</td>
<td>95</td>
<td>squareMagnitude</td>
<td></td>
</tr>
<tr>
<td>FDDlib::ComplexVector</td>
<td>101</td>
<td>~ComplexVector</td>
<td>101</td>
</tr>
<tr>
<td>conjugate</td>
<td>102</td>
<td>conjugate</td>
<td></td>
</tr>
<tr>
<td>deallocate</td>
<td>101</td>
<td>deallocate</td>
<td></td>
</tr>
<tr>
<td>dot</td>
<td>103</td>
<td>dot</td>
<td></td>
</tr>
<tr>
<td>getImag</td>
<td>102</td>
<td>getImag</td>
<td></td>
</tr>
<tr>
<td>getReal</td>
<td>102</td>
<td>getReal</td>
<td></td>
</tr>
<tr>
<td>init</td>
<td>101</td>
<td>init</td>
<td></td>
</tr>
<tr>
<td>len</td>
<td>103</td>
<td>len</td>
<td></td>
</tr>
<tr>
<td>norm</td>
<td>103</td>
<td>norm</td>
<td></td>
</tr>
<tr>
<td>operator()</td>
<td>101</td>
<td>operator()</td>
<td></td>
</tr>
<tr>
<td>size</td>
<td>102</td>
<td>size</td>
<td></td>
</tr>
<tr>
<td>val</td>
<td>101</td>
<td>val</td>
<td></td>
</tr>
<tr>
<td>valVec</td>
<td>103</td>
<td>valVec</td>
<td></td>
</tr>
<tr>
<td>FDDlib::ComplexVector</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complexVector</td>
<td>103</td>
<td>complexVector</td>
<td></td>
</tr>
<tr>
<td>operator*</td>
<td>103</td>
<td>operator*</td>
<td></td>
</tr>
<tr>
<td>operator*</td>
<td>104</td>
<td>operator*</td>
<td></td>
</tr>
<tr>
<td>operator+=</td>
<td>104</td>
<td>operator+=</td>
<td></td>
</tr>
<tr>
<td>operator-</td>
<td>104</td>
<td>operator-</td>
<td></td>
</tr>
<tr>
<td>operator-=</td>
<td>105</td>
<td>operator-=</td>
<td></td>
</tr>
<tr>
<td>operator=</td>
<td>105</td>
<td>operator=</td>
<td></td>
</tr>
<tr>
<td>set</td>
<td>105</td>
<td>set</td>
<td></td>
</tr>
<tr>
<td>setImag</td>
<td>105</td>
<td>setImag</td>
<td></td>
</tr>
<tr>
<td>setReal</td>
<td>106</td>
<td>setReal</td>
<td></td>
</tr>
<tr>
<td>val</td>
<td>106</td>
<td>val</td>
<td></td>
</tr>
<tr>
<td>FDDlib::DenseMatrix</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DenseMatrix</td>
<td>110</td>
<td>DenseMatrix</td>
<td></td>
</tr>
<tr>
<td>lsolve</td>
<td>113</td>
<td>lsolve</td>
<td></td>
</tr>
<tr>
<td>qrdecomp</td>
<td>113</td>
<td>qrdecomp</td>
<td></td>
</tr>
<tr>
<td>FDDlib::Detector2D</td>
<td>114</td>
<td>~Detector2D</td>
<td></td>
</tr>
<tr>
<td>getData</td>
<td>114</td>
<td>getData</td>
<td></td>
</tr>
<tr>
<td>setData</td>
<td>115</td>
<td>setData</td>
<td></td>
</tr>
<tr>
<td>FDDlib::Detector3D</td>
<td>116</td>
<td>~Detector3D</td>
<td></td>
</tr>
<tr>
<td>getData</td>
<td>116</td>
<td>getData</td>
<td></td>
</tr>
<tr>
<td>setData</td>
<td>117</td>
<td>setData</td>
<td></td>
</tr>
<tr>
<td>FDDlib::DiagonalPreconditioner</td>
<td>118</td>
<td>~DiagonalPreconditioner</td>
<td></td>
</tr>
<tr>
<td>DiagonalPreconditioner</td>
<td>119</td>
<td>DiagonalPreconditioner</td>
<td></td>
</tr>
<tr>
<td>FDDlib::DiagonalPreconditioner</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FDDlib::DipoleDetector2D</td>
<td>120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data1_</td>
<td>121</td>
<td>data1_</td>
<td></td>
</tr>
<tr>
<td>data2_</td>
<td>121</td>
<td>data2_</td>
<td></td>
</tr>
<tr>
<td>getData</td>
<td>121</td>
<td>getData</td>
<td></td>
</tr>
<tr>
<td>location1_</td>
<td>121</td>
<td>location1_</td>
<td></td>
</tr>
<tr>
<td>location2_</td>
<td>121</td>
<td>location2_</td>
<td></td>
</tr>
<tr>
<td>weight1_</td>
<td>121</td>
<td>weight1_</td>
<td></td>
</tr>
<tr>
<td>weight2_</td>
<td>121</td>
<td>weight2_</td>
<td></td>
</tr>
</tbody>
</table>

Generated on Mon Aug 30 15:41:16 2004 for FDDLib by Doxygen
bconds, 150
copySubMatrices, 149
currFreq, 150
currSrcNum, 150
grid, 150
knowns, 150
numUnknowns, 150
OMITVAL, 151
Phi, 150
postProcessAMatrix, 149
startTime, 150
stopTime, 150
UNKNOWN_VALUE, 151
updateRightHandSide, 149
usedIterations, 150
FDDlib::FiniteDifferences2D
AdjointSolve, 151
buildAMatrix, 152
FiniteDifferences2D, 151
getAdjointPhi, 152
getOffDiagVal, 153
getPhi, 153
getSourceInfo, 154
getStatistics, 154
MultipleSrcSolve, 154
resetDataVector, 155
rowMap, 157
SingleSrcSolve, 155
solve, 156
validAndActive, 156
FDDlib::FiniteDifferences2D::s
neighbordata, 158
n_linearIndex, 158
n_node, 158
FDDlib::FiniteDifferences3D, 159
achieved Tol, 161
Adep, 161
AdjointPhi, 161
Amat, 161
bconds, 161
copySubMatrices, 160
currFreq, 161
currSrcNum, 161
grid, 161
knowns, 161
numUnknowns, 161
OMITVAL, 162
Phi, 161
postProcessAMatrix, 160
startTime, 161
stopTime, 161
UNKNOWN_VALUE, 162
updateRightHandSide, 160
usedIterations, 161
FDDlib::FiniteDifferences3D
AdjointSolve, 162
buildAMatrix, 163
FiniteDifferences3D, 162
getAdjointPhi, 163
getOffDiagVal, 164
getPhi, 164
ggetSourceInfo, 165
getStatistics, 165
MultipleSrcSolve, 165
resetDataVector, 166
rowMap, 168
SingleSrcSolve, 166
solve, 167
validAndActive, 167
FDDlib::FiniteDifferences3D::s
neighbordata, 169
n_linearIndex, 169
n_node, 169
FDDlib::ILU0Preconditioner, 170
~ILU0Preconditioner, 170
ILU0Preconditioner, 170
init, 171
lowerSolve, 171
solve, 171
upperSolve, 171
FDDlib::Location2D, 173
Location2D, 173, 174
operator-, 174
operator==, 174
setCoordinates, 174
x, 173
y, 173
FDDlib::Location3D, 176
Location3D, 176, 177
operator-, 177
operator==, 177
setCoordinates, 178
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>176</td>
</tr>
<tr>
<td>y</td>
<td>176</td>
</tr>
<tr>
<td>z</td>
<td>176</td>
</tr>
<tr>
<td>FDDlib::Matrix</td>
<td>179</td>
</tr>
<tr>
<td>deallocate</td>
<td>179</td>
</tr>
<tr>
<td>getColumn</td>
<td>179</td>
</tr>
<tr>
<td>getNumCols</td>
<td>179</td>
</tr>
<tr>
<td>getNumRows</td>
<td>179</td>
</tr>
<tr>
<td>getRow</td>
<td>179</td>
</tr>
<tr>
<td>set</td>
<td>180</td>
</tr>
<tr>
<td>val</td>
<td>179</td>
</tr>
<tr>
<td>FDDlib::MonopoleDetector2D</td>
<td>181</td>
</tr>
<tr>
<td>data</td>
<td>182</td>
</tr>
<tr>
<td>getData</td>
<td>181</td>
</tr>
<tr>
<td>getLocation</td>
<td>181</td>
</tr>
<tr>
<td>location</td>
<td>181</td>
</tr>
<tr>
<td>FDDlib::MonopoleDetector2D</td>
<td>182</td>
</tr>
<tr>
<td>MonopoleDetector2D</td>
<td>182</td>
</tr>
<tr>
<td>setData</td>
<td>182</td>
</tr>
<tr>
<td>setLocation</td>
<td>183</td>
</tr>
<tr>
<td>FDDlib::MonopoleDetector3D</td>
<td>184</td>
</tr>
<tr>
<td>data</td>
<td>185</td>
</tr>
<tr>
<td>getData</td>
<td>184</td>
</tr>
<tr>
<td>getLocation</td>
<td>184</td>
</tr>
<tr>
<td>location</td>
<td>184</td>
</tr>
<tr>
<td>FDDlib::MonopoleDetector3D</td>
<td>185</td>
</tr>
<tr>
<td>MonopoleDetector3D</td>
<td>185</td>
</tr>
<tr>
<td>setData</td>
<td>185</td>
</tr>
<tr>
<td>setLocation</td>
<td>186</td>
</tr>
<tr>
<td>FDDlib::MonopoleSource2D</td>
<td>187</td>
</tr>
<tr>
<td>getIntensity</td>
<td>188</td>
</tr>
<tr>
<td>getLocation</td>
<td>187</td>
</tr>
<tr>
<td>getPhase</td>
<td>187</td>
</tr>
<tr>
<td>intensity</td>
<td>188</td>
</tr>
<tr>
<td>location</td>
<td>188</td>
</tr>
<tr>
<td>phase</td>
<td>188</td>
</tr>
<tr>
<td>setIntensity</td>
<td>187</td>
</tr>
<tr>
<td>unsetPhase</td>
<td>187</td>
</tr>
<tr>
<td>FDDlib::MonopoleSource2D</td>
<td>189</td>
</tr>
<tr>
<td>getRightHandSide</td>
<td>189</td>
</tr>
<tr>
<td>MonopoleSource2D</td>
<td>189</td>
</tr>
<tr>
<td>set</td>
<td>189</td>
</tr>
<tr>
<td>setLocation</td>
<td>190</td>
</tr>
<tr>
<td>FDDlib::MonopoleSource3D</td>
<td>191</td>
</tr>
<tr>
<td>getIntensity</td>
<td>192</td>
</tr>
<tr>
<td>getLocation</td>
<td>191</td>
</tr>
<tr>
<td>getPhase</td>
<td>191</td>
</tr>
<tr>
<td>intensity</td>
<td>192</td>
</tr>
<tr>
<td>location</td>
<td>192</td>
</tr>
<tr>
<td>phase</td>
<td>192</td>
</tr>
<tr>
<td>setIntensity</td>
<td>191</td>
</tr>
<tr>
<td>unsetPhase</td>
<td>191</td>
</tr>
</tbody>
</table>

FDDlib::Preconditioner
- init | 226 |
- len | 226 |
- Preconditioner | 226 |
- solve | 227 |
FDDlib::Property
- active | 230 |
- g | 229 |
- getKSquare | 229 |
- getSigma | 229 |
- isActive | 230 |
- mu_a | 231 |
- mu_s | 231 |
- parseArgs | 230 |
- Property | 229 |
- setActive | 230 |
FDDlib::RealVector
- absmax | 233 |
- deallocate | 232 |
- dot | 233 |
- init | 232 |
- len | 234 |
- max | 233 |
- min | 234 |
- norm | 233 |
- norm1 | 233 |
- operator * | 233 |
- operator *= | 233 |
- operator() | 233 |
- operator+ | 233 |
setCenter, 257
setLength, 257
SquareAnomaly, 256
FDDlib::SSORPreconditioner, 258
~SSORPreconditioner, 258
diagonalMultiply, 259
init, 259
lowerSolve, 259
solve, 260
SSORPreconditioner, 258
upperSolve, 260
FDDparse, 18
FDDparse::Parser, 195
acg, 202
arbitrary_detector, 201
arbitrary_prop_path, 201
arbitrary_sources, 201
bg_type, 203
boundaries, 200
circular_anomalies, 200
circular_prop_name, 202
contiguousBoundary, 200
cubical_anomalies, 200
currDetId, 201
currSrcId, 201
deblank, 198
dipole_detector, 201
dipole_sources, 201
domain, 200
domain, 200
domains, 200
equipotential_engines, 196
file, 200
filename, 200
generateArbitraryCartesianGridData, 196
generateArbitraryDetectorData, 196
generateArbitraryPropPath, 197
generateBackgroundType, 196
generateBGLayers, 197
generateBoundaryData, 197
generateCircularAnomalyData, 197
generateConstantBGPropName, 197
generateContiguousBoundaryData, 197
generateCubicalAnomalyData, 197
generateDipoleDetectorData, 196
generateDomain, 195
generateEllipsoidalAnomalyData, 197
generateFilename, 196
generateGridType, 196
generateMapfilePath, 197
generateMonopoleDetectorData, 196
generateMonopoleSourceData, 196
generateNebulousGridData, 196
getNormalEstTol, 198
getNormalNumNeighbors, 197
getNormalPolyOrder, 197
generatePreconditionerData, 197
generatePropertyData, 196
generateRegularCartesianGridData, 196
generateSolveMaxIterations, 198
generateSolveResidualTolerance, 198
generateSolverType, 198
generateSphericalAnomalyData, 197
generateSquareAnomalyData, 197
generateTagType, 198
generateGridType, 202
isContiguouslyBound, 200
layers, 201
mapfile_path, 201
monopole_detector, 201
monopole_sources, 201
num_normalOrder, 202
num_normalTol, 202
num_numNeighbors, 202
ng, 202
nodes, 201
parseAnomaliesTag, 199
parseArbitraryDetector, 199
parseArbitrarySource, 199
parseBCType, 199
parseBoundariesTag, 199
parseDeltaFile, 200
parseDeltaListTag, 200
parseDetectorsTag, 199
parseDipoleDetector, 199
parseDipoleSource, 198
parsedTags, 202
parseFile, 195
parseGridSide, 198
getDipoleSourceData, 196
getDomain, 195
getEllipsoidalAnomalyData, 197
getFilename, 196
getGridType, 196
getMapfilePath, 197
getMonopoleDetectorData, 196
getMonopoleSourceData, 196
getNebulousGridData, 196
getNormalEstTol, 198
getNormalNumNeighbors, 197
getNormalPolyOrder, 197
getPreconditionerData, 197
getPropertyData, 196
getRegularCartesianGridData, 196
generateSolveMaxIterations, 198
generateSolveResidualTolerance, 198
generateSolverType, 198
generateSphericalAnomalyData, 197
generateSquareAnomalyData, 197
generateTagType, 198
generateGridType, 202
isContiguouslyBound, 200
layers, 201
mapfile_path, 201
monopole_detector, 201
monopole_sources, 201
num_normalOrder, 202
num_normalTol, 202
num_numNeighbors, 202
ng, 202
nodes, 201
parseAnomaliesTag, 199
parseArbitraryDetector, 199
parseArbitrarySource, 199
parseBCType, 199
parseBoundariesTag, 199
parseDeltaFile, 200
parseDeltaListTag, 200
parseDetectorsTag, 199
parseDipoleDetector, 199
parseDipoleSource, 198
parsedTags, 202
parseFile, 195
parseGridSide, 198
parseGridTag, 198
parseMediumSetupTag, 199
parseMonopoleDetector, 199
parseMonopoleSource, 198
parseNodeListTag, 199
parseNodeTag, 200
parseOptionsTag, 199
parsePreconditionerType, 199
parsePropertiesTag, 199
Parser, 203
parseSeries, 203
parseSolverType, 199
parseSourcesTag, 198
props, 202
queryIsContiguouslyBound, 197
querySuccessfulParse, 196
reg, 202
solver_conditionertype, 202
solver_iterations, 202
solver_residualtol, 202
solver_type, 202
spherical_anomalies, 200
square_anomalies, 201
FDDparse::Parser::s_arbcartgrid, 205
deltafile, 205
deltax, 205
deltay, 205
deltaz, 205
espeed, 205
xdim, 205
ydim, 205
zdim, 205
FDDparse::Parser::s_arbitrarydetector, 207
id, 207
weight, 207
x, 207
y, 207
z, 207
FDDparse::Parser::s_arbitrarysource, 208
frequencies, 208
id, 208
val, 208
x, 208
y, 208
z, 208
FDDparse::Parser::s_boundary, 209
bt, 209
mixedval, 209
FDDparse::Parser::s_circularanomaly, 210
prop, 210
radius, 210
xcenter, 210
ycenter, 210
FDDparse::Parser::s_cubicalanomaly, 211
length, 211
prop, 211
xcenter, 211
ycenter, 211
zcenter, 211
FDDparse::Parser::s_dipoledetector, 212
id, 212
weight, 212
x, 212
y, 212
z, 212
FDDparse::Parser::s_dipolesource, 213
frequencies, 213
id, 213
magnitude, 213
phase, 213
x, 213
y, 213
z, 213
FDDparse::Parser::s_ellipsoidanomaly, 214
prop, 214
theta1, 214
theta2, 214
theta3, 214
xcenter, 214
xlen, 214
ycenter, 214
ylen, 214
zcenter, 214
zlen, 214
FDDparse::Parser::s_layer, 216
layerpro dbname, 216
layerzdepth, 216
FDDparse::Parser::s_monopoledetector, 217
   id, 217
   xloc, 217
   yloc, 217
   zloc, 217
FDDparse::Parser::s_monopolesource, 218
   frequencies, 218
   id, 218
   magnitude, 218
   phase, 218
   xloc, 218
   yloc, 218
   zloc, 218
FDDparse::Parser::s_nebulousgrid, 219
   espeed, 219
   nodefile, 219
   numnodes, 219
FDDparse::Parser::s_node, 220
   bcondition, 220
   id, 220
   neighbors, 220
   property, 220
   xloc, 220
   yloc, 220
   zloc, 220
FDDparse::Parser::s_precon, 221
   relaxation, 221
   type, 221
FDDparse::Parser::s_property, 222
   active, 222
   coeffs, 222
   name, 222
FDDparse::Parser::s_regcartgrid, 223
   espeed, 223
   stepsizes, 223
   xdim, 223
   ydim, 223
   zdim, 223
FDDparse::Parser::s_sphericalanomaly, 224
   prop, 224
   radius, 224
   xcenter, 224
   ycenter, 224
   zcenter, 224
FDDparse::Parser::s_squareanomaly, 225
   length, 225
   prop, 225
   xcenter, 225
   ycenter, 225
   file
   FDDparse::Parser, 200
   filename
   FDDparse::Parser, 200
   fillUniformProperty
   FDDlib::ArbitraryCartesianGrid2D, 32
   FDDlib::ArbitraryCartesianGrid3D, 42
FiniteDifferences2D
   FDDlib::FiniteDifferences2D, 151
FiniteDifferences3D
   FDDlib::FiniteDifferences3D, 162
frequencies
   FDDlib::Source2D, 242
   FDDlib::Source3D, 245
   FDDparse::Parser::s_arbitrarysource, 208
   FDDparse::Parser::s_dipolesource, 213
   FDDparse::Parser::s_monopolesource, 218
   FDDlib::Property, 229
generateLinearIndices
   FDDlib::ArbitraryCartesianGrid2D, 32
   FDDlib::ArbitraryCartesianGrid3D, 38
getA
   FDDlib::BCMixed2D, 65
INDEX

FDDlib::BCMixed3D, 69
FDDlib::FiniteDifferences2D, 152
FDDlib::FiniteDifferences3D, 163
getAdjointPhi
FDDlib::FiniteDifferences2D, 69
FDDlib::FiniteDifferences3D, 163
getAllBorders
FD-
Dlib::ArbitraryCartesianGrid2D, 28
FD-
Dlib::ArbitraryCartesianGrid3D, 37
getApertureSize
FDDlib::ArbitraryDetector2D, 45
FDDlib::ArbitraryDetector3D, 49
getArbitraryCartesianGridData
FDDparse::Parser, 196
getArbitraryDetectorData
FDDparse::Parser, 196
getArbitraryPropPath
FDDparse::Parser, 197
getArbitrarySourceData
FDDparse::Parser, 196
getBackgroundType
FDDparse::Parser, 196
getBGLayers
FDDparse::Parser, 197
getBorderBySide
FDDlib::ArbitraryCartesianGrid2D, 32
FDDlib::ArbitraryCartesianGrid3D, 42
getBoundaryData
FDDparse::Parser, 197
getCenter
FDDlib::CircularAnomaly, 91
FDDlib::CubicalAnomaly, 107
FDDlib::EllipsoidalAnomaly, 144
FDDlib::SphericalAnomaly, 252
FDDlib::SquareAnomaly, 255
getCircularAnomalyData
FDDparse::Parser, 197
getCol
FDDlib::SparseRowMatrix, 249
column
FDDlib::DenseMatrix, 111
FDDlib::Matrix, 179
FDDlib::SparseRowMatrix, 249
column
FDDlib::DenseMatrix, 111
FDDlib::Matrix, 179
FDDlib::SparseRowMatrix, 249
column
FDDlib::DenseMatrix, 111
FDDlib::Matrix, 179
FDDlib::SparseRowMatrix, 249
getConstantBGPropName
FDDparse::Parser, 197
getContiguousBoundaryData
FDDparse::Parser, 197
cubicalAnomalyData
FDDparse::Parser, 197
data
FDDlib::ArbitraryDetector2D, 46
FDDlib::ArbitraryDetector3D, 50
FDDlib::Detector2D, 114, 115
FDDlib::Detector3D, 116, 117
FDDlib::DipoleDetector2D, 121
FDDlib::DipoleDetector2D, 122
FDDlib::DipoleDetector3D, 126
FDDlib::DipoleDetector3D, 127
FDDlib::MonopoleDetector2D, 181
FDDlib::MonopoleDetector2D, 182
FDDlib::MonopoleDetector3D, 184
FDDlib::MonopoleDetector3D, 185
dipoleDetectorData
FDDparse::Parser, 196
dipoleSourceData
FDDparse::Parser, 196
domain
FDDparse::Parser, 195
dxList
FD-
Dlib::ArbitraryCartesianGrid2D, 27
FD-
Dlib::ArbitraryCartesianGrid3D, 36
dyList
FD-
Dlib::ArbitraryCartesianGrid2D, 27
FD-
Dlib::ArbitraryCartesianGrid3D,
getDzList
FD-Dlib::ArbitraryCartesianGrid3D, 36
getEllipsoidalAnomalyData
FDDparse::Parser, 197
getEnergySpeed
FD-Dlib::ArbitraryCartesianGrid2D, 28
FD-Dlib::ArbitraryCartesianGrid3D, 38
getEntriesPerRow
FDDlib::SparseRowMatrix, 249
getEntry
FDDlib::ArbitraryDetector2D, 47
FDDlib::ArbitraryDetector3D, 51
FDDlib::ArbitrarySource2D, 54
FDDlib::ArbitrarySource3D, 58
getEulerAngles
FDDlib::EllipsoidalAnomaly, 144
ggetFieldSize
FDDlib::ArbitrarySource2D, 53
FDDlib::ArbitrarySource3D, 57
getFilename
FDDparse::Parser, 196
getFrequencies
FDDlib::Source2D, 241
FDDlib::Source3D, 244
getFrequency
FDDlib::Source2D, 242
FDDlib::Source3D, 245
getGridType
FDDparse::Parser, 196
getImag
FDDlib::Complex, 94
FDDlib::ComplexVector, 102
getIntensities
FDDlib::DipoleSource2D, 133
FDDlib::DipoleSource3D, 139
ggetIntensity
FDDlib::MonopoleSource2D, 188
FDDlib::MonopoleSource3D, 192
getKSquare
FDDlib::Property, 229
ggetLength
FDDlib::CubicalAnomaly, 107
FDDlib::SquareAnomaly, 255
ggetLengths
FDDlib::EllipsoidalAnomaly, 144
ggetLinearIndex
FDDlib::CartesianNode2D, 84
FDDlib::CartesianNode3D, 89
ggetLocation
FDDlib::CartesianNode2D, 81
FDDlib::CartesianNode3D, 86
FDDlib::MonopoleDetector2D, 181
FDDlib::MonopoleDetector3D, 184
FDDlib::MonopoleSource2D, 187
FDDlib::MonopoleSource3D, 191
ggetLocations
FDDlib::DipoleDetector2D, 122
FDDlib::DipoleDetector3D, 127
FDDlib::DipoleSource2D, 133
FDDlib::DipoleSource3D, 139
ggetMapfilePath
FDDparse::Parser, 197
ggetMonopoleDetectorData
FDDparse::Parser, 196
ggetMonopoleSourceData
FDDparse::Parser, 196
ggetNebulousGridData
FDDparse::Parser, 196
ggetNeighbor
FDDlib::CartesianNode2D, 84
FDDlib::CartesianNode3D, 89
ggetNode
FD-Dlib::ArbitraryCartesianGrid2D, 28
FD-Dlib::ArbitraryCartesianGrid3D, 38
INDEX

37
getNormalEstTol
  FDDparse::Parser, 198
getNormalNumNeighbors
  FDDparse::Parser, 197
getNormalPolyOrder
  FDDparse::Parser, 197
getNormalVector
  FDDlib::CartesianNode2D, 81
  FDDlib::CartesianNode3D, 86
getNumCols
  FDDlib::DenseMatrix, 111
  FDDlib::Matrix, 179
  FDDlib::SparseRowMatrix, 249
getNumFrequencies
  FDDlib::Source2D, 241
  FDDlib::Source3D, 244
getNumNeighbors
  FDDlib::CartesianNode2D, 81
  FDDlib::CartesianNode3D, 86
getNumRows
  FDDlib::DenseMatrix, 111
  FDDlib::Matrix, 179
  FDDlib::SparseRowMatrix, 249
getOffDiagVal
  FDDlib::FiniteDifferences2D, 153
  FDDlib::FiniteDifferences3D, 164
getPhase
  FDDlib::MonopoleSource2D, 187
  FDDlib::MonopoleSource3D, 191
getPhases
  FDDlib::DipoleSource2D, 133
  FDDlib::DipoleSource3D, 139
getPhi
  FDDlib::FiniteDifferences2D, 153
  FDDlib::FiniteDifferences3D, 164
getPoles
  FDDlib::DipoleSource2D, 133
  FDDlib::DipoleSource3D, 139
getPreconditionerData
  FDDparse::Parser, 198
getProperty
  FDDlib::Anomaly2D, 22
  FDDlib::Anomaly3D, 24
  FDDlib::CartesianNode2D, 81
  FDDlib::CartesianNode3D, 86
getPropertyData
  FDDparse::Parser, 196
getRadius
  FDDlib::CircularAnomaly, 91
  FDDlib::SphericalAnomaly, 252
getReal
  FDDlib::Complex, 94
  FDDlib::ComplexVector, 102
getRegularCartesianGridData
  FDDparse::Parser, 196
getRightHandSide
  FDDlib::ArbitrarySource2D, 55
  FDDlib::ArbitrarySource3D, 59
  FDDlib::DipoleSource2D, 134
  FDDlib::DipoleSource3D, 140
  FDDlib::MonopoleSource2D, 189
  FDDlib::MonopoleSource3D, 193
  FDDlib::Source2D, 243
  FDDlib::Source3D, 246
ggetRow
  FDDlib::DenseMatrix, 111
  FDDlib::Matrix, 179
  FDDlib::SparseRowMatrix, 249
ggetRowEntry
  FDDlib::SparseRowMatrix, 249
getSigma
  FDDlib::Property, 228
getAddressMaxIterations
  FDDparse::Parser, 198
getAddressResidualTolerance
  FDDparse::Parser, 198
getAddressSolverType
  FDDparse::Parser, 198
getAddressSourceInfo
  FDDlib::FiniteDifferences2D, 154
  FDDlib::FiniteDifferences3D, 165

Generated on Mon Aug 30 15:41:16 2004 for FDDLib by Doxygen
getSphericalAnomalyData
    FDDparse::Parser, 197
getSquareAnomalyData
    FDDparse::Parser, 197
getStatistics
    FDDlib::FiniteDifferences2D, 154
    FDDlib::FiniteDifferences3D, 165
getStep
    FDDlib::RegularCartesianGrid2D, 236
    FDDlib::RegularCartesianGrid3D, 238
getTagType
    FDDparse::Parser, 198
getWeights
    FDDlib::DipoleDetector2D, 122
    FDDlib::DipoleDetector3D, 127
grid
    FDDlib::FiniteDifferences2D, 150
    FDDlib::FiniteDifferences3D, 161
gridType
    FDDparse::Parser, 202
id
    FDDparse::Parser::s_arbitrarydetector, 207
    FDDparse::Parser::s_arbitrarysource, 208
    FDDparse::Parser::s_dipoledetector, 212
    FDDparse::Parser::s_dipolesource, 213
    FDDparse::Parser::s_monopoledetector, 217
    FDDparse::Parser::s_monopolesource, 218
    FDDparse::Parser::s_node, 220
imagPart_
    FDDlib::Complex, 96
INACTIVE
    FDDlib::CartesianNode2D, 82
    FDDlib::CartesianNode3D, 87
init
    FDDlib::ComplexVector, 101
    FDDlib::DenseMatrix, 113
    FDDlib::DiagonalPreconditioner, 119
    FDDlib::ILU0Preconditioner, 171
    FDDlib::Preconditioner, 227
    FDDlib::RealVector, 232
    FDDlib::SparseRowMatrix, 248
    FDDlib::SSORPreconditioner, 259
insertAnomaly
    FDDlib::ArbitraryCartesianGrid2D, 32
    FDDlib::ArbitraryCartesianGrid3D, 42
int1_
    FDDlib::DipoleSource2D, 131
    FDDlib::DipoleSource3D, 137
int2_
    FDDlib::DipoleSource2D, 131
    FDDlib::DipoleSource3D, 137
intensity_
    FDDlib::MonopoleSource2D, 188
    FDDlib::MonopoleSource3D, 192
isActive
    FDDlib::Property, 230
isBorderNode
    FDDlib::CartesianNode2D, 82
    FDDlib::CartesianNode3D, 87
isContiguouslyBound
    FDDparse::Parser, 200
knowns
    FDDlib::FiniteDifferences2D, 150
    FDDlib::FiniteDifferences3D, 161
INDEX 281

layerpropname
  FDDparse::Parser::s_layer, 216
layers
  FDDparse::Parser, 201
layerzdepth
  FDDparse::Parser::s_layer, 216
len_
  FDDlib::ComplexVector, 103
  FDDlib::CubicalAnomaly, 108
  FDDlib::Preconditioner, 226
  FDDlib::RealVector, 234
  FDDlib::SquareAnomaly, 256
length
  FDDparse::Parser::s_cubicalanomaly, 211
  FDDparse::Parser::s_squareanomaly, 225
linearIndex_
  FDDlib::CartesianNode2D, 82
  FDDlib::CartesianNode3D, 87
location
  FDDlib::ArbitraryDetector2D::s_adentry, 48
  FDDlib::ArbitraryDetector2D, 121
  FDDlib::ArbitraryDetector3D::s_adentry, 52
  FDDlib::ArbitraryDetector3D, 126
  FDDlib::ArbitrarySource2D::s_asentry, 56
  FDDlib::ArbitrarySource2D, 131
  FDDlib::ArbitrarySource3D::s_asentry, 60
  FDDlib::ArbitrarySource3D, 137
location1_
  FDDlib::DipoleDetector2D, 121
  FDDlib::DipoleDetector3D, 126
  FDDlib::DipoleDetector3D::s_boundary, 181
  FDDlib::DipoleDetector3D, 184
  FDDlib::DipoleDetector3D::s_monopolesource, 188
  FDDlib::DipoleDetector3D, 192
lowerSolve
  FDDlib::ILU0Preconditioner, 171
  FDDlib::SSORPreconditioner, 259
LSSolve
  FDDlib::DenseMatrix, 113
magnitude
  FDDlib::Complex, 95
  FDDparse::Parser::s_dipolesource, 213
  FDDparse::Parser::s_monopolesource, 218
mapfile_path
  FDDparse::Parser, 201
max
  FDDlib::RealVector, 233
  FDDlib::SparseRowMatrix, 250
  FDDlib::SparseRowMatrix, 250
min
  FDDlib::RealVector, 234
mixedval
  FDDparse::Parser::s_boundary, 209
monopole_detectors
  FDDparse::Parser, 201
monopole_sources
  FDDparse::Parser, 201
MonopoleDetector2D
  FDDlib::MonopoleDetector2D, 182
MonopoleDetector3D
  FDDlib::MonopoleDetector3D, 185
MonopoleSource2D

Generated on Mon Aug 30 15:41:16 2004 for FDDLib by Doxygen
FDDlib::MonopoleSource2D, 188, 189
MonopoleSource3D
  FDDlib::MonopoleSource3D, 192, 193
mu_a
  FDDlib::Property, 231
mu_s
  FDDlib::Property, 231
MultipleSrcSolve
  FDDlib::FiniteDifferences2D, 154
  FDDlib::FiniteDifferences3D, 165
n_borderNode
  FDDlib::BCMixed2D::s_neighborData, 68
  FDDlib::BCMixed3D::s_neighborData, 72
n_linearIndex
  FDDlib::BCMixed2D::s_neighborData, 68
  FDDlib::BCMixed3D::s_neighborData, 72
  FDDlib::FiniteDifferences2D::s_neighborData, 158
  FDDlib::FiniteDifferences3D::s_neighborData, 169
n_node
  FDDlib::BCMixed2D::s_neighborData, 68
  FDDlib::BCMixed3D::s_neighborData, 72
  FDDlib::FiniteDifferences2D::s_neighborData, 158
  FDDlib::FiniteDifferences3D::s_neighborData, 169
n_normalOrder
  FDDparse::Parser, 202
n_normalTol
  FDDparse::Parser, 202
n_numNeighbors
  FDDparse::Parser, 202
name
  FDDparse::Parser::s_property, 222
neighbors
  FDDparse::Parser::s_node, 220
neighbors
  FDDlib::CartesianNode2D, 82
  FDDlib::CartesianNode3D, 87
ng
  FDDparse::Parser, 202
nodefile
  FDDparse::Parser::s_nebulousgrid, 219
nodeList
  FDlib::ArbitraryCartesianGrid2D, 29
  FDlib::ArbitraryCartesianGrid3D, 39
nodes
  FDDparse::Parser, 201
nodes
  FDDlib::BoundaryCondition2D, 76
  FDDlib::BoundaryCondition3D, 79
norm
  FDDlib, 16
  FDDlib::ComplexVector, 103
  FDDlib::RealVector, 233
norm1
  FDDlib, 16
  FDDlib::RealVector, 233
normalVector
  FDDlib::CartesianNode2D, 82
  FDDlib::CartesianNode3D, 87
numActiveNodes
  FDlib::ArbitraryCartesianGrid2D, 29
  FDlib::ArbitraryCartesianGrid3D, 38
numActiveNodes
  FDlib::ArbitraryCartesianGrid2D, 29
  FDlib::ArbitraryCartesianGrid3D, 38
operator+
    FDDlib::Complex, 98
    FDDlib::ComplexVector, 104
    FDDlib::DenseMatrix, 111
    FDDlib::RealVector, 233
operator+=
    FDDlib::Complex, 98
    FDDlib::ComplexVector, 104
    FDDlib::RealVector, 232
operator-
    FDDlib::Complex, 98
    FDDlib::ComplexVector, 104
    FDDlib::DenseMatrix, 111
    FDDlib::Location2D, 174
    FDDlib::Location3D, 177
    FDDlib::RealVector, 233
operator-=
    FDDlib::Complex, 98
    FDDlib::ComplexVector, 105
    FDDlib::RealVector, 232
operator/
    FDDlib::Complex, 99
operator/= 
    FDDlib::Complex, 99
operator/= 
    FDDlib::Complex, 100
    FDDlib::ComplexVector, 105
    FDDlib::DenseMatrix, 110
    FDDlib::RealVector, 232
    FDDlib::SparseRowMatrix, 248
operator==
    FDDlib::Location2D, 174
    FDDlib::Location3D, 177
parseAnomaliesTag 
    FDDparse::Parser, 199
parseArbitraryDetector 
    FDDparse::Parser, 199
parseArbitrarySource 
    FDDparse::Parser, 199
parseArgs
    FDDlib::Property, 230
parseBCType
    FDDparse::Parser, 199
parseBoundariesTag 
    FDDparse::Parser, 199
parseDeltaFile
    FDDparse::Parser, 200
parseDeltaListTag 
    FDDparse::Parser, 200
parseDetectorsTag
    FDDparse::Parser, 199
parseDipoleDetector
    FDDparse::Parser, 199
parseDipoleSource
    FDDparse::Parser, 198
parsedTags
    FDDparse::Parser, 202
parseFile
    FDDparse::Parser, 195
parseGridSide 
    FDDparse::Parser, 198
parseGridTag
    FDDparse::Parser, 198
parseMediumSetupTag 
    FDDparse::Parser, 199
parseMonopoleDetector
    FDDparse::Parser, 199
parseMonopoleSource
    FDDparse::Parser, 199
parseNodeListTag
    FDDparse::Parser, 198
parseNodeTag
    FDDparse::Parser, 200
parseOptionsTag
    FDDparse::Parser, 199
parsePreconditionerType 
    FDDparse::Parser, 199
parsePropertiesTag 
    FDDparse::Parser, 199
Parser
    FDDparse::Parser, 203
parseSeries 
    FDDparse::Parser, 203
parseSolverType
    FDDparse::Parser, 199
parseSourcesTag 
    FDDparse::Parser, 198
phase
    FDDparse::Parser::s- 
    dipolesource, 213
INDEX

FDDlib::DenseMatrix, 112

set
FDDlib::ComplexVector, 105
FDDlib::DenseMatrix, 111
FDDlib::DipoleSource2D, 134
FDDlib::DipoleSource3D, 140
FDlib::DynamicSparseRowMatrix, 142
FDDlib::Matrix, 180
FDDlib::MonopoleSource2D, 189
FDDlib::MonopoleSource3D, 193
FDDlib::RealVector, 233
FDDlib::SparseRowMatrix, 250

setA
FDDlib::BCMixed2D, 65
FDDlib::BCMixed3D, 69

setActive
FDDlib::Property, 230

setCenter
FDDlib::CircularAnomaly, 93
FDDlib::CubicalAnomaly, 109
FDDlib::EllipsoidalAnomaly, 144
FDDlib::SphericalAnomaly, 254
FDDlib::SquareAnomaly, 257

setCoordinates
FDDlib::Location2D, 174
FDDlib::Location3D, 178

setData
FDDlib::ArbitraryDetector2D, 47
FDDlib::ArbitraryDetector3D, 51
FDDlib::Detector2D, 115
FDDlib::Detector3D, 117
FDDlib::DipoleDetector2D, 123
FDDlib::DipoleDetector3D, 128
FDDlib::MonopoleDetector2D, 182
FDDlib::MonopoleDetector3D, 185

setDelta
FDDlib::ArbitraryCartesianGrid2D, 33
FDDlib::ArbitraryCartesianGrid3D, 43

setEnergySpeed
FDlib::ArbitraryCartesianGrid2D, 28
FDlib::ArbitraryCartesianGrid3D, 38

setEntriesPerRow
FDDlib::DynamicSparseRowMatrix, 143

setEulerAngles
FDDlib::EllipsoidalAnomaly, 147

setFrequencies
FDDlib::Source2D, 243
FDDlib::Source3D, 246

setImag
FDDlib::Complex, 94
FDDlib::ComplexVector, 105

setIntensities
FDDlib::DipoleSource2D, 134
FDDlib::DipoleSource3D, 140

setIntensity
FDDlib::MonopoleSource2D, 187
FDDlib::MonopoleSource3D, 191

setLength
FDDlib::CubicalAnomaly, 109
FDDlib::SquareAnomaly, 257

setLengths
FDDlib::EllipsoidalAnomaly, 147

setLinearIndex
FDDlib::CartesianNode2D, 84
FDDlib::CartesianNode3D, 89

setLocation
FDDlib::MonopoleDetector2D, 183
FDDlib::MonopoleDetector3D, 186
FDDlib::MonopoleSource2D, 190
INDEX

FDDlib::MonopoleSource3D, 194
setLocations
FDDlib::DipoleDetector2D, 123
FDDlib::DipoleDetector3D, 128
FDDlib::DipoleSource2D, 135
FDDlib::DipoleSource3D, 141
setNeighbor
FDDlib::CartesianNode2D, 85
FDDlib::CartesianNode3D, 90
setNormalVector
FDDlib::CartesianNode2D, 85
FDDlib::CartesianNode3D, 90
setPhase
FDDlib::MonopoleSource2D, 187
FDDlib::MonopoleSource3D, 191
setPhases
FDDlib::DipoleSource2D, 135
FDDlib::DipoleSource3D, 141
setProperty
FDDlib::Anomaly2D, 23
FDDlib::Anomaly3D, 25
FDDlib::CartesianNode2D, 85
FDDlib::CartesianNode3D, 90
setRadius
FDDlib::CircularAnomaly, 93
FDDlib::SphericalAnomaly, 254
setReal
FDDlib::Complex, 94
FDDlib::ComplexVector, 106
setStep
FDDlib::RegularCartesianGrid2D, 237
FDDlib::RegularCartesianGrid3D, 239
setupNodes
FD-
Dlib::ArbitraryCartesianGrid2D, 29
FD-
Dlib::ArbitraryCartesianGrid3D, 38
setWeights
FDDlib::DipoleDetector2D, 123
FDDlib::DipoleDetector3D, 128
SingleSrcSolve
FDDlib::FiniteDifferences2D, 155
FDDlib::FiniteDifferences3D, 166
size
FDDlib::ComplexVector, 102
FDDlib::RealVector, 233
solve
FDDlib::DiagonalPreconditioner, 119
FDDlib::FiniteDifferences2D, 156
FDDlib::FiniteDifferences3D, 167
FDDlib::ILU0Preconditioner, 171
FDDlib::Preconditioner, 227
FDDlib::SSORPreconditioner, 260
solver_conditionertype
FDDparse::Parser, 202
solver_iterations
FDDparse::Parser, 202
solver_residualtol
FDDparse::Parser, 202
solver_type
FDDparse::Parser, 202
Source2D
FDDlib::Source2D, 241, 242
Source3D
FDDlib::Source3D, 244, 245
SparseRowMatrix
FDDlib::SparseRowMatrix, 248
spherical_anomalies
FDDparse::Parser, 200
SphericalAnomaly
FDDlib::SphericalAnomaly, 253
square_anomalies
FDDparse::Parser, 201
SquareAnomaly
FDDlib::SquareAnomaly, 256
squareMagnitude
FDDlib::Complex, 95
SSORPreconditioner
FDDlib::SSORPreconditioner, 258
startTime
FDDlib::FiniteDifferences2D, 150
FDDlib::FiniteDifferences3D, 161
step
FD-
Dlib::RegularCartesianGrid2D, 236
FD-
Dlib::RegularCartesianGrid3D, 239
stepsize
FDDparse::Parser::s_regcartgrid, 223
stopTime
FDDlib::FiniteDifferences2D, 150
FDDlib::FiniteDifferences3D, 161
theta1
FDDparse::Parser::s_ellipsoidalanomaly, 214
theta1
FDDlib::EllipsoidalAnomaly, 145
theta2
FDDparse::Parser::s_ellipsoidalanomaly, 214
theta2
FDDlib::EllipsoidalAnomaly, 145
theta3
FDDparse::Parser::s_ellipsoidalanomaly, 214
theta3
FDDlib::EllipsoidalAnomaly, 145
transpose
FDDlib::DenseMatrix, 111
type
FDDparse::Parser::s_precon, 221
U
FDDlib::EllipsoidalAnomaly, 145
UD_{neg2U}^\top
FDDlib::EllipsoidalAnomaly, 145
UNKNOWN\_LIN\_INDEX
FDDlib::CartesianNode2D, 83
FDDlib::CartesianNode3D, 88
UNKNOWN\_NUMACTIVE
FD-
Dlib::ArbitraryCartesianGrid2D, 30
FD-
Dlib::ArbitraryCartesianGrid3D, 39
UNKNOWN\_VALUE
FDDlib::BCMixed2D, 66
FDDlib::BCMixed3D, 70
FDDlib::FiniteDifferences2D, 151
FDDlib::FiniteDifferences3D, 162
updateData
FDDlib::EllipsoidalAnomaly, 145
updateRightHandSide
FDDlib::FiniteDifferences2D, 149
FDDlib::FiniteDifferences3D, 160
upperSolve
FDDlib::ILU0Preconditioner, 171
FDDlib::SSORPreconditioner, 260
usedIterations
FDDlib::FiniteDifferences2D, 150
FDDlib::FiniteDifferences3D, 161
val
FDDlib::ComplexVector, 101
FDDlib::ComplexVector, 106
FDDlib::DenseMatrix, 111

Generated on Mon Aug 30 15:41:16 2004 for FDDLib by Doxygen
FDDlib::Matrix, 179
FDDlib::RealVector, 233
FDDlib::SparseRowEntry, 247
FDDlib::SparseRowMatrix, 251
FDDparse::Parser::s_arbitrarysource, 208
validAndActive
FDDlib::FiniteDifferences2D, 156
FDDlib::FiniteDifferences3D, 167
value
FDDlib::ArbitrarySource2D::s_asentry, 56
FDDlib::ArbitrarySource3D::s_asentry, 60
valVec
FDDlib::ComplexVector, 103
FDDlib::DenseMatrix, 112
FDDlib::RealVector, 234
weight
FDDlib::ArbitraryDetector2D::s_adentry, 48
FDDlib::ArbitraryDetector3D::s_adentry, 52
FDDparse::Parser::s_arbitrarydetector, 207
FDDparse::Parser::s_dipoledetector, 212
weight1
FDDlib::DipoleDetector2D, 121
FDDlib::DipoleDetector3D, 126
weight2
FDDlib::DipoleDetector2D, 121
FDDlib::DipoleDetector3D, 126
x
FDDlib::Location2D, 173
FDDlib::Location3D, 176
FDDparse::Parser::s_arbitrarydetector, 207
FDDparse::Parser::s_arbitrarysource, 208
FDDparse::Parser::s_dipoledetector, 212
FDDparse::Parser::s_dipolesource, 213
xcenter
FDDparse::Parser::s_arbitrarysource, 208
circularanomaly, 210
FDDparse::Parser::s_cubicalanomaly, 211
FDDparse::Parser::s_ellipsoidalanomaly, 214
FDDparse::Parser::s_sphericalanomaly, 224
FDDparse::Parser::s_squaresource, 225
xdim
FDDparse::Parser::s_arbcartgrid, 205
FDDparse::Parser::s_regcartgrid, 223
xlen
FDDparse::Parser::s_arbitradydetector, 207
FDDparse::Parser::s_dipoledetector, 212
ellipsoidalanomaly, 214
xlen
FDDlib::EllipsoidalAnomaly, 145
xloc
FDDparse::Parser::s_monopoledetector, 217
FDDparse::Parser::s_monopolesource, 218
FDDparse::Parser::s_node, 220
xoffset
FDDlib::ArbitraryCartesianGrid2D::s_offsets, 35
FDDlib::ArbitraryCartesianGrid3D::s_offsets, 44
y
FDDlib::Location2D, 173
FDDlib::Location3D, 176
FDDparse::Parser::s_arbitrarydetector, 207
INDEX

FDDparse::Parser::s_arbitrarysource, 208
FDDparse::Parser::s_dipoledetector, 212
FDDparse::Parser::s_dipolesource, 213

ycenter
FDDparse::Parser::s_circularanomaly, 210
FDDparse::Parser::s_cubicalanomaly, 211
FDDparse::Parser::s_ellipsoidalanomaly, 214
FDDparse::Parser::s_sphericalanomaly, 224
FDDparse::Parser::s_squareanomaly, 225

ydim
FDDparse::Parser::s_arbcartgrid, 205
FDDparse::Parser::s_regcartgrid, 223

ylen
FDDparse::Parser::s_ellipsoidalanomaly, 214

FDDlib::EllipsoidalAnomaly, 145

ylen_
FDDlib::EllipsoidalAnomaly, 145

yloc
FDDparse::Parser::s_monopoledetector, 217
FDDparse::Parser::s_monopolesource, 218
FDDparse::Parser::s_node, 220

zoffset
FD-
Dlib::ArbitraryCartesianGrid2D::s_offsets, 35
FD-
Dlib::ArbitraryCartesianGrid3D::s_offsets, 44

z
FDDlib::Location3D, 176
FDDparse::Parser::s_arbitrarydetector, 207

zcenter
FDDparse::Parser::s_cubicalanomaly, 211
FDDparse::Parser::s_ellipsoidalanomaly, 214
FDDparse::Parser::s_sphericalanomaly, 224

zdim
FDDparse::Parser::s_arbcartgrid, 205
FDDparse::Parser::s_regcartgrid, 223

zlen
FDDparse::Parser::s_ellipsoidalanomaly, 214

zlen_
FDDlib::EllipsoidalAnomaly, 145

zloc
FDDparse::Parser::s_monopoledetector, 217
FDDparse::Parser::s_monopolesource, 218
FDDparse::Parser::s_node, 220

zoffset
FD-
Dlib::ArbitraryCartesianGrid3D::s_offsets, 44

Generated on Mon Aug 30 15:41:16 2004 for FDDLib by Doxygen