

SPARTANS Project Reference Manual

0.01

Sun Oct 28 20:49:22 2007

Contents

1 SPARTANS Project Namespace Documentation	1
2 SPARTANS Project Data Structure Documentation	1
3 SPARTANS Project File Documentation	21

1 SPARTANS Project Namespace Documentation

1.1 std Namespace Reference

2 SPARTANS Project Data Structure Documentation

2.1 Boundary Class Reference

[Boundary](#) Class defined by its identifiant, nodes number and id nodes vector.

```
#include <boundary.h>
```

Public Member Functions

- [Boundary](#) ()
Constructor.
- virtual [~Boundary](#) ()
DistructOr.

Data Fields

- int [id](#)
Boundary identifiant.
- int * [idnode](#)
Id nodes vector.
- int [nb_node](#)
Nodes number at the boundary.

2.1.1 Detailed Description

[Boundary](#) Class defined by its identifiant, nodes number and id nodes vector.
Definition at line 18 of file boundary.h.

2.1.2 Constructor & Destructor Documentation

2.1.2.1 Boundary ()

Constructor.

Definition at line 15 of file boundary.cpp.

```
16 {  
17  
18 }
```

2.1.2.2 ~Boundary () [virtual]

DistructOr.

Definition at line 20 of file boundary.cpp.

```
21 {  
22     delete[] idnode;  
23 }
```

2.1.3 Field Documentation

2.1.3.1 int id

[Boundary](#) identifiant.

Definition at line 24 of file boundary.h.

2.1.3.2 int* idnode

Id nodes vector.

Definition at line 26 of file boundary.h.

2.1.3.3 int nb_node

Nodes number at the boundary.

Definition at line 22 of file boundary.h.

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

- [boundary.h](#)
- [boundary.cpp](#)

2.2 Cell Class Reference

[Cell](#) Class defined by its identifiant and its nodes.

```
#include <cell.h>
```

Collaboration diagram for Cell:

Public Member Functions

- [Cell](#) ()
Constructor.
- virtual [~Cell](#) ()
Destructor.

Data Fields

- double [area](#)
Area (2D)/Volume (3D) of the cell.
- [Edge](#) * [cedge](#)
Edges vector.
- [Node](#) [center_node](#)
Centroid node.
- [Face](#) * [cface](#)
Faces vector.
- int [id](#)
Cell Identifiant.
- int [id_adj_cell](#) [10]
Adjacent cells identifiant vector.
- int * [idnode](#)
Id nodes vector.
- int [nb_adj_cell](#)
Number of adjacent cells.
- int [nb_edge](#)
Number of edges.
- int [nb_node](#)
Number of nodes.

2.2.1 Detailed Description

[Cell](#) Class defined by its identifiant and its nodes.

Definition at line 19 of file cell.h.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 Cell ()

Constructor.

Definition at line 15 of file cell.cpp.

```
16 {
17     //nb_node=nnode;
18     //idnode=new int[mnode];
19 }
```

2.2.2.2 ~Cell () [virtual]

Destructor.

Definition at line 21 of file cell.cpp.

```
22 {
23     delete[] idnode;
24     delete[] cedge;
25 }
```

2.2.3 Field Documentation

2.2.3.1 double area

Area (2D)/Volume (3D) of the cell.

Definition at line 37 of file cell.h.

2.2.3.2 Edge* cedge

Edges vector.

Definition at line 33 of file cell.h.

2.2.3.3 Node center_node

Centroid node.

Definition at line 29 of file cell.h.

2.2.3.4 Face* cface

Faces vector.

Definition at line 35 of file cell.h.

2.2.3.5 int id

[Cell](#) Identifient.

Definition at line 27 of file cell.h.

2.2.3.6 int id_adj_cell[10]

Adjacent cells identifient vector.

Definition at line 41 of file cell.h.

2.2.3.7 int* idnode

Id nodes vector.

Definition at line 25 of file cell.h.

2.2.3.8 int nb_adj_cell

Number of adjacent cells.

Definition at line 39 of file cell.h.

2.2.3.9 int nb_edge

Number of edges.

Definition at line 31 of file cell.h.

2.2.3.10 int nb_node

Number of nodes.

Definition at line 23 of file cell.h.

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

- [cell.h](#)
- [cell.cpp](#)

2.3 Edge Class Reference

[Edge](#) Class defined by its identifient and 2 nodes.

```
#include <edge.h>
```

Collaboration diagram for Edge:

Public Member Functions

- [Edge \(\)](#)
Constructor.
- virtual [~Edge \(\)](#)
Destructor.

Data Fields

- char [etype](#)
Edge type : etype='i' in [Edge](#) (default), etype='o' out [Edge](#).
- int [id](#)
Edge identifiant.
- int [id_adj_cell](#) [10]
Adjacent cells identifiant vector.
- double [length](#)
Edge length.
- [Node](#) [mid_edge_node](#)
Mid edge node.
- int [nb_adj_cell](#)
Adjacent cells number.
- [Node](#) [node1](#)
1st node of the [Edge](#).
- [Node](#) [node2](#)
2nd node of the [Edge](#).

2.3.1 Detailed Description

[Edge](#) Class defined by its identifiant and 2 nodes.

Definition at line 19 of file [edge.h](#).

2.3.2 Constructor & Destructor Documentation

2.3.2.1 [Edge \(\)](#) [inline]

Constructor.

Definition at line 39 of file [edge.h](#).

```
39 {etype='i'};
```

2.3.2.2 `~Edge ()` [virtual]

Destructor.

Definition at line 16 of file edge.cpp.

```
17 {  
18     //dtor  
19 }
```

2.3.3 Field Documentation

2.3.3.1 `char etype`

[Edge](#) type : etype='i' in [Edge](#) (default), etype='o' out [Edge](#).

Definition at line 37 of file edge.h.

2.3.3.2 `int id`

[Edge](#) identifiant.

Definition at line 23 of file edge.h.

2.3.3.3 `int id_adj_cell[10]`

Adjacent cells identifiant vector.

Definition at line 35 of file edge.h.

2.3.3.4 `double length`

[Edge](#) length.

Definition at line 31 of file edge.h.

2.3.3.5 `Node mid_edge_node`

Mid edge node.

Definition at line 29 of file edge.h.

2.3.3.6 `int nb_adj_cell`

Adjacent cells number.

Definition at line 33 of file edge.h.

2.3.3.7 Node node1

1st node of the [Edge](#).

Definition at line 25 of file edge.h.

2.3.3.8 Node node2

2nd node of the [Edge](#).

Definition at line 27 of file edge.h.

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

- [edge.h](#)
- [edge.cpp](#)

2.4 Face Class Reference

[Face](#) Class defined by its identifent and its nodes.

```
#include <face.h>
```

Collaboration diagram for Face:

Public Member Functions

- [Face](#) ()
Constructor.
- virtual [~Face](#) ()
Destructor.

Data Fields

- [Node](#) * [fnode](#)
Nodes vector.
- int [id](#)
Face identifiant.
- int [nb_node](#)
Nodes number.

2.4.1 Detailed Description

[Face](#) Class defined by its identifent and its nodes.

Definition at line 17 of file face.h.

2.4.2 Constructor & Destructor Documentation

2.4.2.1 Face ()

Constructor.

Definition at line 16 of file face.cpp.

```
17 {  
18     //ctor  
19 }
```

2.4.2.2 ~Face () [virtual]

Destructor.

Definition at line 21 of file face.cpp.

```
22 {  
23     //dtor  
24 }
```

2.4.3 Field Documentation

2.4.3.1 Node* fnode

Nodes vector.

Definition at line 25 of file face.h.

2.4.3.2 int id

Face identifiant.

Definition at line 21 of file face.h.

2.4.3.3 int nb_node

Nodes number.

Definition at line 23 of file face.h.

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

- [face.h](#)
- [face.cpp](#)

2.5 Grid Class Reference

[Grid](#) Class defined by its nodes, cells and boundaries.

```
#include <grid.h>
```

Collaboration diagram for Grid:

Public Member Functions

- void [FindAdjacentCells](#) ()
Identify the cell's and edge's adjacent cells.
- [Grid](#) ()
Constructor.
- void [LoadBoundariesData](#) ()
Build boundaries.
- void [LoadCellsData](#) ()
Build Cells, centroid nodes, mid edges nodes, cells area, edges length,...
- void [LoadGlobalData](#) ()
Define global parameters of the [Grid](#).
- void [LoadNodesData](#) ()
Build the nodes vectors.
- void [WriteGridReport](#) ()
Generate Grid's report.
- virtual [~Grid](#) ()
Destructor.

Data Fields

- int [dim](#)
Space dimension : 2 for 2D and 3 for 3D.
- [Boundary](#) * [gboundary](#)
Boundaries number.
- [Cell](#) * [gcell](#)
Cells vector.
- [Node](#) * [gnode](#)
Nodes number.
- int [nb_boundary](#)
Boundaries number.
- int [nb_cell](#)
Cells number.

- int `nb_face`
Faces number.
- int `nb_node`
Nodes number.

2.5.1 Detailed Description

`Grid` Class defined by its nodes, cells and boundaries.

Definition at line 26 of file `grid.h`.

2.5.2 Constructor & Destructor Documentation

2.5.2.1 `Grid ()` [inline]

Constructor.

Definition at line 46 of file `grid.h`.

```
46 {nb_cell=0; nb_node=0; nb_boundary=0; nb_face=0;};
```

2.5.2.2 `~Grid ()` [virtual]

Destructor.

Definition at line 16 of file `grid.cpp`.

```
17 {  
18     delete[] gcell;  
19     delete[] gnode;  
20     delete[] gboundary;  
21 }
```

2.5.3 Member Function Documentation

2.5.3.1 `void FindAdjacentCells ()`

Identify the cell's and edge's adjacent cells.

Definition at line 219 of file `grid.cpp`.

```
220 {  
221     for(int i=0;i<nb_cell;i++)  
222     {  
223         gcell[i].nb_adj_cell=0;  
224  
225         for(int k=0;k<gcell[i].nb_edge;k++)  
226         {  
227             for(int j=0;j<nb_cell;j++)
```

```

228     {
229         if(i!=j)
230         { //gcell[i].cedge[k].nb_adj_cell=0;
231           //gcell[i].cedge[k].id_adj_cell[0]=gcell[i].id;
232           for(int m=0;m<gcell[j].nb_edge;m++)
233           {
234             if(gcell[i].cedge[k].node1.id==gcell[j].cedge[m].node2.id
235               &&gcell[i].cedge[k].node2.id==gcell[j].cedge[m].node1.id
236               ||gcell[i].cedge[k].node1.id==gcell[j].cedge[m].node1.id
237               &&gcell[i].cedge[k].node2.id==gcell[j].cedge[m].node2.id)
238             {
239               gcell[i].id_adj_cell[gcell[i].nb_adj_cell]=gcell[j].id;
240
241               gcell[i].cedge[k].id_adj_cell[gcell[i].cedge[k].nb_adj_cell]=gcell[j].id;
242
243               //gcell[i].cedge[j].id_adj_cell[0]
244               gcell[i].nb_adj_cell++;
245               gcell[i].cedge[k].nb_adj_cell++;
246             }
247           }
248         }
249     }
250 }
251 }
252 }

```

2.5.3.2 void LoadBoundariesData ()

Build boundaries.

Definition at line 187 of file grid.cpp.

```

188 {
189     char* filename="tmp\\boundaries.tmp";
190     ifstream infile(filename);
191     int tmp;
192     string ctmp;
193
194     for(int i=0;i<nb_boundary;i++)
195     {
196         infile>>ctmp;
197
198         gboundary[i].id=i+1;
199
200         infile>>tmp;
201         infile>>gboundary[i].nb_node;
202         infile>>tmp;
203         infile>>tmp;
204
205         gboundary[i].idnode=new int[gboundary[i].nb_node];
206
207         for(int j=0;j<gboundary[i].nb_node;j++)
208         {
209             infile>>gboundary[i].idnode[j];
210             gnode[gboundary[i].idnode[j]-1].ntype='o';
211         }
212
213     }
214 }
215 }

```

```

216     infile.close();
217 }

```

2.5.3.3 void LoadCellsData ()

Build Cells, centroid nodes, mid edges nodes, cells area, edges length,...

Definition at line 69 of file grid.cpp.

```

70 {
71     char* filename="tmp\\cells.tmp";
72     ifstream infile(filename);
73     int tmp;
74     int nd1, nd2;
75     double x1,x2,y1,y2,z1,z2;
76
77     for(int i=0;i<nb_cell;i++)
78     {
79         // read cells data
80         infile>>gcell[i].id;
81         infile>>tmp;
82         infile>>gcell[i].nb_node;
83
84         gcell[i].idnode=new int[gcell[i].nb_node];
85         gcell[i].cedge=new Edge[gcell[i].nb_node];
86
87         gcell[i].center_node.id=gcell[i].id;
88         gcell[i].center_node.x=0;
89         gcell[i].center_node.y=0;
90         gcell[i].center_node.z=0;
91
92         for(int j=0;j<gcell[i].nb_node;j++)
93         {
94             infile>>gcell[i].idnode[j];
95             gcell[i].cedge[j].id=j;
96             // centro97          gcell[i].center_node.x+=gnode[gcell[i].idnode[j]-1].x;
97             gcell[i].center_node.y+=gnode[gcell[i].idnode[j]-1].y;
98             gcell[i].center_node.z+=gnode[gcell[i].idnode[j]-1].z;
99         }
100
101         gcell[i].center_node.x/=gcell[i].nb_node;
102         gcell[i].center_node.y/=gcell[i].nb_node;
103         gcell[i].center_node.z/=gcell[i].nb_node;
104
105         // compute edges properties
106         nd1=gcell[i].idnode[0]-1;
107
108         x1=gnode[nd1].x;
109         y1=gnode[nd1].y;
110         z1=gnode[nd1].z;
111         gcell[i].nb_edge=gcell[i].nb_node;
112         for(int j=0;j<gcell[i].nb_edge-1;j++)
113         {
114             nd2=gcell[i].idnode[j+1]-1;
115
116             // edge nodes
117             gcell[i].cedge[j].node1=gnode[nd1];
118             gcell[i].cedge[j].node2=gnode[nd2];
119             //edge type 'i' in edge, 'o' out edge
120             if(gcell[i].cedge[j].node1.ntype=='o'&&gcell[i].cedge[j].node2.ntype=='o')

```

```

123     {
124         gcell[i].cedge[j].etype='o';
125     }
126
127     //id of the first adjacent cell of the edge
128     gcell[i].cedge[j].id_adj_cell[0]=gcell[i].id;
129
130
131     x2=gnode[nd2].x;
132     y2=gnode[nd2].y;
133     z2=gnode[nd2].z;
134
135     // compute mid edges nodes
136     gcell[i].cedge[j].mid_edge_node.x=(x1+x2)/2;
137     gcell[i].cedge[j].mid_edge_node.y=(y1+y2)/2;
138     gcell[i].cedge[j].mid_edge_node.z=(z1+z2)/2;
139
140     // compute edge length;
141     gcell[i].cedge[j].length=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)+(z1-z2)*(z1-z2));
142
143
144     nd1=nd2;
145     x1=x2;
146     y1=y2;
147     z1=z2;
148 }
149
150     nd1=gcell[i].idnode[0]-1;
151     x1=gnode[nd1].x;
152     y1=gnode[nd1].y;
153     z1=gnode[nd1].z;
154
155     nd2=gcell[i].idnode[gcell[i].nb_node-1]-1;
156     x2=gnode[nd2].x;
157     y2=gnode[nd2].y;
158     z2=gnode[nd2].z;
159
160     gcell[i].cedge[gcell[i].nb_node-1].node1=gnode[nd1];
161     gcell[i].cedge[gcell[i].nb_node-1].node2=gnode[nd2];
162
163     gcell[i].cedge[gcell[i].nb_node-1].mid_edge_node.x=(x1+x2)/2;
164     gcell[i].cedge[gcell[i].nb_node-1].mid_edge_node.y=(y1+y2)/2;
165     gcell[i].cedge[gcell[i].nb_node-1].mid_edge_node.z=(z1+z2)/2;
166
167     gcell[i].cedge[gcell[i].nb_node-1].length=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)+(z1-z2)*(z1-z2));
168
169     // area of triangular cell : Heron formula
170     // -----
171     // area= $\sqrt{p(p-a)(p-b)(p-c)}$  where a,b and c are the cell edges length, p = (a+b+c)/2 the half perimeter
172
173     double p=0,a,b,c;
174     if(gcell[i].nb_edge==3)
175     {
176         a=gcell[i].cedge[0].length;
177         b=gcell[i].cedge[1].length;
178         c=gcell[i].cedge[2].length;
179         p=(a+b+c)/2;
180         gcell[i].area=sqrt(p*(p-a)*(p-b)*(p-c));
181     }
182 }
183
184     infile.close();
185 }

```

2.5.3.4 void LoadGlobalData ()

Define global parameters of the [Grid](#).

Definition at line 23 of file grid.cpp.

```
24 {
25     int ncell, nnode, nboundary;
26     char* filename="tmp\\global.tmp";
27     ifstream infile(filename);
28
29     string tmp;
30
31     infile>>tmp;
32     infile>>tmp;
33     infile>>tmp;
34     infile>>tmp;
35     infile>>tmp;
36     infile>>tmp;
37
38     infile>>nb_node;
39     infile>>nb_cell;
40     infile>>tmp;
41     infile>>nb_boundary;
42     infile>>dim;
43     infile>>tmp;
44
45
46     gcell=new Cell[nb_cell];
47     gnode=new Node[nb_node];
48     gboundary=new Boundary[nb_boundary];
49
50
51     infile.close();
52 }
```

2.5.3.5 void LoadNodesData ()

Build the nodes vectors.

Definition at line 54 of file grid.cpp.

```
55 {
56     char* filename="tmp\\nodes.tmp";
57     ifstream infile(filename);
58     for(int i=0;i<nb_node;i++)
59     {
60         infile>>gnode[i].id;
61         infile>>gnode[i].x;
62         infile>>gnode[i].y;
63         if(dim==2) gnode[i].z=0;
64         if(dim==3) infile>>gnode[i].z;
65     }
66     infile.close();
67 }
```

2.5.3.6 void WriteGridReport ()

Generate Grid's report.

Definition at line 255 of file grid.cpp.

```

256 {
257     char* filename="tmp\\report.tmp";
258     ofstream outfile(filename);
259
260
261     outfile<<"nb_node "<<nb_node<<endl;
262     outfile<<"nb_cell "<<nb_cell<<endl;
263     outfile<<"nb_boundary "<<nb_boundary<<endl;
264     outfile<<"dim "<<dim<<endl<<endl;
265
266
267
268
269     for(int i=0;i<nb_cell;i++)
270     {
271         outfile<<"cell "<<gcell[i].id<<endl<<endl;
272         outfile<<"\t nb of adjacent cells : "<<gcell[i].nb_adj_cell<<endl;
273         outfile<<"\t adjacent cells : ";
274         for(int k=0;k<gcell[i].nb_adj_cell;k++)
275         {
276             outfile<<gcell[i].id_adj_cell[k]<<" ";
277         }
278         outfile<<endl;
279         outfile<<"\t area : "<<gcell[i].area<<endl;
280         outfile<<"\tcenter node : "<<"\t\t";
281         outfile<<gcell[i].center_node.x<<"\t\t";
282         outfile<<gcell[i].center_node.y<<"\t\t";
283         outfile<<gcell[i].center_node.z<<endl;
284
285         for(int j=0;j<gcell[i].nb_node;j++)
286         {
287             outfile<<"\tnode "<<gcell[i].idnode[j]<<" \t:\t";
288             outfile<<gnode[gcell[i].idnode[j]-1].x<<"\t\t";
289             outfile<<gnode[gcell[i].idnode[j]-1].y<<"\t\t";
290             outfile<<gnode[gcell[i].idnode[j]-1].z<<endl;
291
292             outfile<<"\tedge "<<gcell[i].cedge[j].id<<endl;
293             outfile<<"\t\t type : "<<gcell[i].cedge[j].etype<<endl;
294             outfile<<"\t\t nb adjacent cells : "<<gcell[i].cedge[j].nb_adj_cell<<endl;
295             outfile<<"\t\t adjacent cells : ";
296             for(int m=0;m<gcell[i].cedge[j].nb_adj_cell;m++)
297             {
298                 outfile<<gcell[i].cedge[j].id_adj_cell[m]<<" ";
299             }
300             outfile<<endl;
301
302             outfile<<"\tnodes : "<<gcell[i].cedge[j].node1.id<<" "<<gcell[i].cedge[j].node1.ntyep<<" "<<gcel
303             outfile<<"\tlength : "<<gcell[i].cedge[j].length<<endl;
304             outfile<<"\tmid edge node : "<<gcell[i].cedge[j].mid_edge_node.x
305                 <<"\t\t"<<gcell[i].cedge[j].mid_edge_node.y
306                 <<"\t\t"<<gcell[i].cedge[j].mid_edge_node.z;
307             outfile<<endl;
308         }
309         outfile<<"-----"<<endl;
310     }
311     outfile.close();
312     cout<<"end!";
313 }

```

2.5.4 Field Documentation

2.5.4.1 `int dim`

Space dimension : 2 for 2D and 3 for 3D.

Definition at line 38 of file grid.h.

2.5.4.2 `Boundary* gboundary`

Boundaries number.

Definition at line 44 of file grid.h.

2.5.4.3 `Cell* gcell`

Cells vector.

Definition at line 40 of file grid.h.

2.5.4.4 `Node* gnode`

Nodes number.

Definition at line 42 of file grid.h.

2.5.4.5 `int nb_boundary`

Boundaries number.

Definition at line 34 of file grid.h.

2.5.4.6 `int nb_cell`

Cells number.

Definition at line 30 of file grid.h.

2.5.4.7 `int nb_face`

Faces number.

Definition at line 36 of file grid.h.

2.5.4.8 `int nb_node`

Nodes number.

Definition at line 32 of file grid.h.

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

- [grid.h](#)
- [grid.cpp](#)

2.6 Node Class Reference

`Node` Class defined by its coordinates, identifiant and type.

```
#include <node.h>
```

Public Member Functions

- `Node ()`
Constructor (default values of coordinates are set 0 and the node type is set 'i').
- `virtual ~Node ()`
Destructor.

Data Fields

- `int id`
Node identifiant.
- `char ntype`
Node type : ntype='i' for in nodes, ntype='o' for out nodes (boundaries node).
- `double x`
Nodes coordinate, z=0 in 2D configuration.
- `double y`
- `double z`

2.6.1 Detailed Description

`Node` Class defined by its coordinates, identifiant and type.

Definition at line 16 of file node.h.

2.6.2 Constructor & Destructor Documentation

2.6.2.1 `Node ()` [inline]

Constructor (default values of coordinates are set 0 and the node type is set 'i').

Definition at line 26 of file node.h.

```
26 {x=0;y=0;z=0;ntype='i'};
```

2.6.2.2 `~Node ()` [virtual]

Destructor.

Definition at line 16 of file node.cpp.

```
17 {  
18     //dtor  
19 }
```

2.6.3 Field Documentation

2.6.3.1 `int id`

[Node](#) identifiant.

Definition at line 24 of file node.h.

2.6.3.2 `char ntype`

[Node](#) type : ntype='i' for in nodes, ntype='o' for out nodes (boundaries node).

Definition at line 22 of file node.h.

2.6.3.3 `double x`

Nodes coordinate, z=0 in 2D configuration.

Definition at line 20 of file node.h.

2.6.3.4 `double y`

Definition at line 20 of file node.h.

2.6.3.5 `double z`

Definition at line 20 of file node.h.

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

- [node.h](#)
- [node.cpp](#)

3 SPARTANS Project File Documentation

3.1 boundary.cpp File Reference

```
#include "boundary.h"
```

Include dependency graph for boundary.cpp:

3.2 boundary.h File Reference

```
#include <tchar.h>
```

```
#include <string>
```

Include dependency graph for boundary.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- class [Boundary](#)
Boundary Class defined by its identifiant, nodes number and id nodes vector.

3.3 cell.cpp File Reference

```
#include "cell.h"
```

Include dependency graph for cell.cpp:

3.4 cell.h File Reference

```
#include "node.h"
```

```
#include "edge.h"
```

```
#include "face.h"
```

Include dependency graph for cell.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- class [Cell](#)
Cell Class defined by its identifiant and its nodes.

3.5 edge.cpp File Reference

```
#include "edge.h"
```

Include dependency graph for edge.cpp:

3.6 edge.h File Reference

```
#include "node.h"
```

Include dependency graph for edge.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- class [Edge](#)
Edge Class defined by its identifiant and 2 nodes.

3.7 face.cpp File Reference

```
#include "face.h"
```

Include dependency graph for face.cpp:

3.8 face.h File Reference

```
#include "node.h"
```

Include dependency graph for face.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- class [Face](#)
Face Class defined by its identifent and its nodes.

3.9 grid.cpp File Reference

```
#include "grid.h"
```

Include dependency graph for grid.cpp:

3.10 grid.h File Reference

```
#include <fstream>
```

```
#include <iostream>
```

```
#include <math.h>
```

```
#include <tchar.h>
```

```
#include <string>
```

```
#include "cell.h"
```

```
#include "node.h"
```

```
#include "boundary.h"
```

Include dependency graph for grid.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- class [Grid](#)
Grid Class defined by its nodes, cells and boundaries.

3.11 main.cpp File Reference

```
#include <iostream>
```

```
#include <conio.h>
```

```
#include "spartans.h"
```

Include dependency graph for main.cpp:

Namespaces

- namespace [std](#)

Functions

- int [main](#) (int argc, char *argv[])
Main function : test program.

3.11.1 Function Documentation

3.11.1.1 int main (int argc, char * argv[])

Main function : test program.

Definition at line 30 of file main.cpp.

```
31 {
32     Grid M;
33
34     LoadGambitData("data\\test_case.neu");
35
36     //cout<<"fin des transferts..."<<endl;
37
38     M.LoadGlobalData();
39     M.LoadNodesData();
40     M.LoadBoundariesData();
41     M.LoadCellsData();
42 }
```

```
43
44     M.FindAdjacentCells();
45
46     M.WriteGridReport();
47
48     getch();
49     return 0;
50 }
```

3.12 node.cpp File Reference

```
#include "node.h"
```

Include dependency graph for node.cpp:

3.13 node.h File Reference

This graph shows which files directly or indirectly include this file:

Data Structures

- class [Node](#)
Node Class defined by its coordinates, identifiant and type.

3.14 spartans.h File Reference

```
#include <fstream>
#include <math.h>
#include <tchar.h>
#include <string>
#include "sources/grid.h"
#include "sources/cell.h"
#include "sources/node.h"
#include "sources/edge.h"
```

Include dependency graph for spartans.h:

This graph shows which files directly or indirectly include this file:

Functions

- void [LoadGambitData](#) (char *FileName)
Load Gambit 2.3.6 neutral file.

3.14.1 Function Documentation

3.14.1.1 void LoadGambitData (char * *FileName*)

Load Gambit 2.3.6 neutral file.

Definition at line 28 of file spartans.h.

```
29 {
30
31     string str="";
32
33     char c;
34     char *sval="",tmp;
35     int nl=0,k=0;
36     bool in_global=false;
37     bool in_nodes=false;
38     bool in_elements=true;
39     bool in_boundaries=true;
40
41     ifstream infile(FileName);
42     //ofstream outfile("tmp\\000.tmp");
43     ofstream outglobal("tmp\\global.tmp");
44     ofstream outnodes("tmp\\nodes.tmp");
45     ofstream outelements("tmp\\cells.tmp");
46     ofstream outboundaries("tmp\\boundaries.tmp");
47
48     if(infile.is_open())
49     {
50         while(!infile.eof())
51         {
52             infile>>str;
53             if(str=="CONTROL")
54             {
55                 do
56                 {
57                     infile>>str;
58                     if(str=="NUMNP") in_global=true;
59                     if(in_global==true&&str!="ENDOFSECTION")outglobal<<str<<endl;
60                 }while(str!="ENDOFSECTION");
61             }
62
63             if(str=="NODAL")
64             {
65                 do
66                 {
67                     infile>>str;
68                     if(str!="COORDINATES" &&
69                        str!="2.2.30" &&
70                        str!="ENDOFSECTION") outnodes<<str<<endl;
71                 }while(str!="ENDOFSECTION");
72             }
73
74             if(str=="ELEMENTS/CELLS")
75             {
76                 do
77                 {
78                     infile>>str;
79                     if(str!="2.2.30" &&
80                        str!="ENDOFSECTION")outelements<<str<<endl;
81                 }while(str!="ENDOFSECTION");
82             }
83         }
84     }
85 }
```

```
83         }
84
85         if(str=="BOUNDARY")
86         {
87             do
88             {
89                 infile>>str;
90                 if(str!="BOUNDARY"&&
91                    str!="CONDITIONS"&&
92                    str!="2.2.30"&&
93                    str!="ENDOFSECTION")outboundaries<<str<<endl;
94             }while(!infile.eof());
95         }
96         //outfile<<str<<endl;
97     }
98 }
99
100     infile.close();
101
102     //outfile.close();
103     outglobal.close();
104     outnodes.close();
105     outelements.close();
106     outboundaries.close();
107
108 }
```

Index

- ~Boundary
 - Boundary, [3](#)
- ~Cell
 - Cell, [4](#)
- ~Edge
 - Edge, [6](#)
- ~Face
 - Face, [7](#)
- ~Grid
 - Grid, [8](#)
- ~Node
 - Node, [13](#)
- area
 - Cell, [4](#)
- Boundary, [2](#)
 - ~Boundary, [3](#)
 - Boundary, [3](#)
 - id, [3](#)
 - idnode, [3](#)
 - nb_node, [3](#)
- boundary.cpp, [14](#)
- boundary.h, [14](#)
- cedge
 - Cell, [4](#)
- Cell, [3](#)
 - ~Cell, [4](#)
 - area, [4](#)
 - cedge, [4](#)
 - Cell, [4](#)
 - center_node, [4](#)
 - cface, [4](#)
 - id, [5](#)
 - idnode, [5](#)
 - nb_node, [5](#)
- cell.cpp, [15](#)
- cell.h, [15](#)
- center_node
 - Cell, [4](#)
- cface
 - Cell, [4](#)
- dim
 - Grid, [12](#)
- Edge, [5](#)
 - ~Edge, [6](#)
 - Edge, [5](#)
 - id, [6](#)
 - length, [6](#)
 - mid_edge_node, [6](#)
 - node1, [6](#)
 - node2, [6](#)
- edge.cpp, [15](#)
- edge.h, [15](#)
- Face, [6](#)
 - ~Face, [7](#)
 - Face, [7](#)
 - fnode, [7](#)
 - id, [7](#)
 - nb_node, [7](#)
- face.cpp, [15](#)
- face.h, [16](#)
- fnode
 - Face, [7](#)
- gboundary
 - Grid, [12](#)
- gcell
 - Grid, [12](#)
- gnode
 - Grid, [12](#)
- Grid, [8](#)
 - ~Grid, [8](#)
 - dim, [12](#)
 - gboundary, [12](#)
 - gcell, [12](#)
 - gnode, [12](#)
 - Grid, [8](#)
 - LoadBoundariesData, [8](#)
 - LoadCellsData, [9](#)
 - LoadGlobalData, [11](#)
 - LoadNodesData, [11](#)
 - nb_boundary, [12](#)
 - nb_cell, [13](#)
 - nb_face, [13](#)
 - nb_node, [13](#)
 - WriteGridReport, [11](#)
- grid.cpp, [16](#)
- grid.h, [16](#)
- id

- Boundary, [3](#)
- Cell, [5](#)
- Edge, [6](#)
- Face, [7](#)
- Node, [14](#)
- idnode
 - Boundary, [3](#)
 - Cell, [5](#)
- length
 - Edge, [6](#)
- LoadBoundariesData
 - Grid, [8](#)
- LoadCellsData
 - Grid, [9](#)
- LoadGambitData
 - spartans.h, [18](#)
- LoadGlobalData
 - Grid, [11](#)
- LoadNodesData
 - Grid, [11](#)
- main
 - main.cpp, [17](#)
- main.cpp, [16](#)
 - main, [17](#)
- mid_edge_node
 - Edge, [6](#)
- nb_boundary
 - Grid, [12](#)
- nb_cell
 - Grid, [13](#)
- nb_face
 - Grid, [13](#)
- nb_node
 - Boundary, [3](#)
 - Cell, [5](#)
 - Face, [7](#)
 - Grid, [13](#)
- Node, [13](#)
 - ~Node, [13](#)
 - id, [14](#)
 - Node, [13](#)
 - ntype, [14](#)
 - x, [14](#)
 - y, [14](#)
 - z, [14](#)
- node.cpp, [17](#)
- node.h, [17](#)
- node1
 - Edge, [6](#)
- node2
 - Edge, [6](#)
- ntype
 - Node, [14](#)
- Répertoire de référence de E:/, [1](#)
- Répertoire de référence de E:/0-
Recherche/, [1](#)
- Répertoire de référence de E:/0-
Recherche/5-SPARTANS_-
Project/, [1](#)
- Répertoire de référence de E:/0-
Recherche/5-SPARTANS_-
Project/spartans_src_pc_-
home/, [2](#)
- Répertoire de référence de E:/0-
Recherche/5-SPARTANS_-
Project/spartans_src_pc_-
home/sources/, [1](#)
- spartans.h, [17](#)
 - LoadGambitData, [18](#)
- std, [2](#)
- WriteGridReport
 - Grid, [11](#)
- x
 - Node, [14](#)
- y
 - Node, [14](#)
- z
 - Node, [14](#)