A PLATFORM THAT ACCEPTS SUB-GRID MODELS AS PLUGINS TO ENABLE THE TESTING OF LES MODELS AGAINST DNS DATA ARCHIVED IN THE JOHNS HOPKINS TURBULENT DATABASE

A means of rapidly evaluating the accuracy of LES models of turbulence against DNS data.

Igor Grossman and Graham Thorpe
College of Engineering & Science
Victoria University, Melbourne, Australia

Version 0.1.1
Initial revision
May 2016

ABSTRACT

The Johns Hopkins Turbulent Databases (JHTDB) is a publicly accessible archive of solutions of the Navier-Stokes equations obtained for forced isotropic turbulence. The solutions are obtained by direct numerical simulation (DNS) of the Navier-Stokes equation and they are accurate to about six decimal places. However, the solution is obtained at 1024*1024*1024 points in space and 1024 time samples that span about one turnover time of the largest flow structure. The entire space-time history of turbulence contains in excess of 1024 data points and it is accessible to users remotely through an interface that is based on the Web-services model. The JHTDB is an invaluable source of data against which models of turbulence can be evaluated. However, this database contains 160 petabytes (1.6×1017 PB) of information and it is a serious challenge if we wish to use it routinely for practical analyses (9,779,997,251,909 points queried). The natural answer to this challenge is to seek the application of database technology. in computational fluid dynamics (CFD) and turbulence research.

The purpose of this work is to create platform based on modular programming that allows LES models to be dynamically loaded and to be rapidly evaluated against data in the JHTDB. The calls diagrams, class lists and source code are provided to help and support code logic and usage of the proposed platform.

Further assistance with using the work can be obtained from Mr Igor Grossman at igor.grossman@live.vu.edu.au

Table of Contents

A PLATFORM THAT ACCEPTS SUB-GRID MODELS AS PLUGINS TO ENABLE THE TESTING OF LES MODELS AGAINST DNS DATA ARCHIVED IN THE JOHNS HOPKINS TURBULENT DATABASE.

A means of rapidly evaluating the accuracy of LES models of turbulence against DNS data.

1 INTRODUCTION

Flow structures in turbulent flows span many orders of magnitude of length and time scales. They range from the length scale at which very small eddies lose their coherence as their translational kinetic energy is dissipated into heat, up to the scale of the eddies, the size of which is related to that of the macroscopic system. The behavior of the range of structures can be captured by assuming that the fluid is a continuum, and they can be described by solving the Navier-Stokes equation. However, the limitations of computers restrict to solutions of the Navier-Stokes equation to low Reynolds number flows in simple geometries. These limitations make it infeasible to resolve features of turbulent flows on the smallest length and time scales.

Engineers and scientists must therefore resort to empirical models of these small scale phenomena that can be expressed in simple mathematical terms. The models typically involve some form of averaging and approximations, hence it is essential that we have some simple way of comparing their accuracy with the exact solutions of the Navier-Stokes equation.

This report presents the source code of software that enables engineers and scientists to develop and evaluate the accuracy of large eddy simulation models. This is achieved by the development of a platform that allows the new models to be compared with DNS solutions. The software is designed in such a way that the new models are introduced as standalone plugins. This renders the software easy to use.

In addition to the software presented in this report users will require access to a number of components, namely cmake, boost, FFTW, sqllite, C++.

Should the reader require assistance to implement this software they should contact Igor Grossman, igor.grossman@gmail.com.

2 BACKGROUND INFORMATION

The Johns Hopkins Turbulent Databases JHTDB is a publicly accessible archive of solutions of the Navier-Stokes equations obtained for forced turbulence. The solutions are obtained by direct numerical simulation (DNS) of the Navier-Stokes equation and they are accurate to about six

significant figures. However, the solution is obtained at 1024*1024*1024 points in space and 1024 time samples that span about one turnover time of the largest flow structure. The entire space-time history of turbulence contains in excess of 1024 data points and it is accessible to users remotely through an interface that is based on the Web-services model. The JHTDB is an invaluable source of data against which models of turbulence can be evaluated.

3 RUNTIME MODEL

Thw Johns Hopkins database of the flow under consideration contains 160 petabytes of information and it is a serious challenge if we wish to use it routinely for practical analyses (9,779,997,251,909 points queried).

An answer to this challenge is to seek the application of database technology to computational fluid dynamics (CFD) and turbulence research. Turbulent flows are inherently unsteady, and this can have significant implications in many practical situations. For example, the formation of wakes may give rise to large fluctuating forces on bluff bodies immersed in turbulent flows, and methods must be found to attenuate these forces. Flows through tree canopies, say, are extremely important in determining the rate of exchange of gases such as carbon dioxide and water vapour with the atmosphere. However, we have noted that DNS solutions are presently unobtainable in most practical applications. The Navier-Stokes equation can be time-averaged but this results in a loss of considerable detail of the flow. However, the behaviour of large eddies can be captured by resorting to the tautologically defined large eddy simulation (LES). As noted above, DNS resolves all of the scales that influence turbulent flows, but in the case of LES the Navier-Stokes equation is spatially filtered so that it is expressed in terms of the velocities of large scale structures. The rate of viscous dissipation is quantified by modelling the shear stress, and it is this process that can lead to error. A means of rapid testing and evaluation of models is therefore required and this involves working with large data sets.

This, in turn, requires the development not only better and more powerful hardware but new paradigms and methodologies in software development. Modular programming techniques have been created to meet these needs. Modular programming is based on the concept of partitioning the functionality of a program into number independent modules. Modules are typically incorporated into a program through interfaces. This approach was subsequently incorporated into object oriented programming, parallelization and the use of plugins.

The purpose of this work is to create platform based on modular programming that allows LES models to be dynamically loaded and to be rapidly evaluated against data in the JHTDB.

The calls diagrams, class lists and source code below are here to help and support the code logic and usage of the proposed platform.

3.1 TARGET PLATFORMS

Project developed and builds using an open source GNU C++ compiler. It is constructed in such a way that it should compile under most C++ ANSI compilers. All examples and source code was build and run on HPC (Edward supercomputer) located in Melbourne University. Makefile was generated using Cmake - open source utility for building and maintaining Linux projects. The project makes use of boost, C++, SQLite and FFTW components .

Notes

The following commands must be executed prior to attempting to build the project:

- module load cmake
- module load boost
- module load sqlite
- module load fftw

3.2 PERFORMANCE

The project has been built, developed, tested and run on the Edward High Performance Cluster based at The University of Melbourne. It executes commands about three orders of magnitude faster than personal computers. It has about six orders more RAM that a personal computer. But its main advantage is that it is based on the GNU/Linux operation system, which renders it compatible with any other flavour of the Linux operation system.

3.3 EXAMPLES

Examples located at DNS_plugins/Example directory on Edward supercomputer. They contain Makefiles and source code for a number of examples and help to enable users to invoke and call utilities of the proposed platform.

All Source code – contact Igor. If they want a tar file.

.

4 NAMESPACE INDEX

4.1 NAMESPACE LIST

Here is a list of all namespaces with brief descriptions:

5 CLASS INDEX

5.1 CLASS HIERARCHY

This inheritance list is sorted more or less, but not completely, alphabetically:

5.2 CLASS LIST

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

6 FILE INDEX

6.1 FILE LIST

Here is a list of all files with brief descriptions:

7 NAMESPACE DOCUMENTATION

7.1 plugins Namespace Reference

7.1.1 Classes

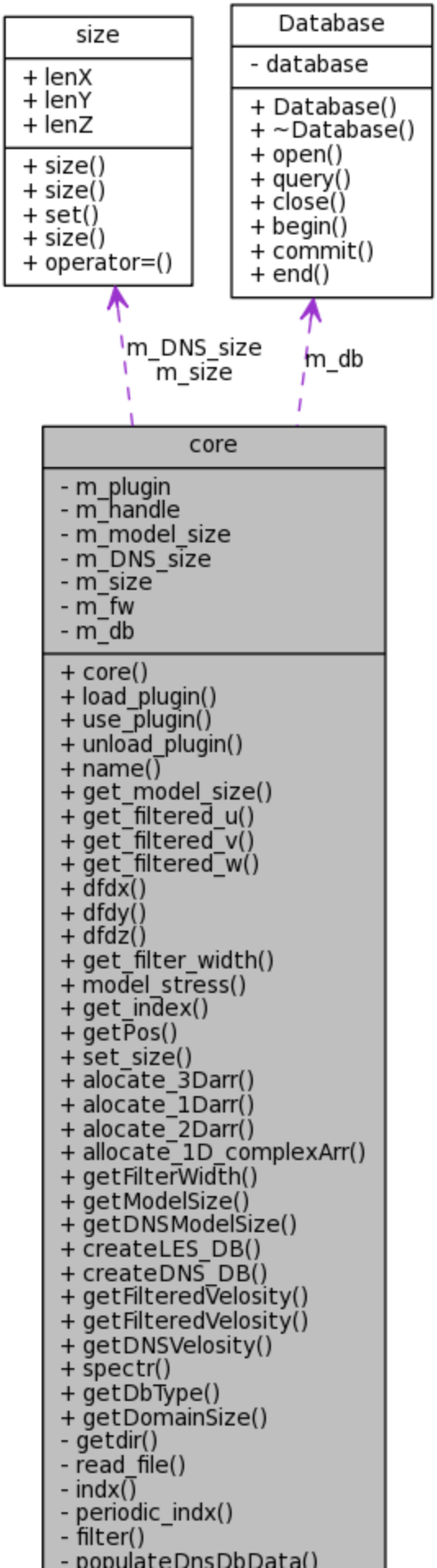
- class Plugin
- class Smagorinsky

8 CLASS DOCUMENTATION

8.1 core Class Reference

#include <core.h>

Collaboration diagram for core:


size
+ lenX
+ lenY
+ lenZ
+ size()
+ size()
+ set()
+ size()
+ operator=()
Database
- database
+ Database()
+ ~Database()
+ open()
+ query()
+ close()
+ begin()
+ commit()
+ end()
m_DNS_size
m_size
m_db
core
- m_plugin
- m_handle
- m_model_size
- m_DNS_size
- m_size
- m_fw
- m_db
+ core()
+ load_plugin()
+ use_plugin()
+ unload_plugin()
+ name()
+ get_model_size()
+ get_filtered_u()
+ get_filtered_v()
+ get_filtered_w()
+ dfdx()
+ dfdy()
+ dfdz()
+ get_filter_width()
+ model_stress()
+ get_index()
+ getPos()
+ set_size()
+ alocate_3Darr()
+ alocate_1Darr()
+ alocate_2Darr()
+ allocate_1D_complexArr()
+ getFilterWidth()
+ getModelSize()
+ getDNSModelSize()
+ createLES_DB()
+ createDNS_DB()
+ getFilteredVelosity()
+ getFilteredVelosity()
+ getDNSVelosity()
+ spectr()
+ getDbType()
+ getDomainSize()
- getdir()
- read_file()
- indx()
- periodic_indx()
- filter()
- populateDnsDbData()

8.1.1 Public Member Functions

- core ()
- void load_plugin ()
- void use_plugin ()
- void unload_plugin ()
- std::string name ()
- int get_model_size ()
- double * get_filtered_u ()
- double * get_filtered_v ()
- double * get_filtered_w ()
- double dfdx (int ind, order_t o, double *u)
- double dfdy (int ind, order_t o, double *u)
- double dfdz (int ind, order_t o, double *u)
- int get_filter_width ()
- void model_stress ()
- int get_index (size &pos)
- size getPos ()
- void set_size (size &s)
- double * alocate_3Darr (size &)
- double * alocate_1Darr (int N)
- double * alocate_2Darr (int Nx, int Ny)
- fftw_complex * allocate_1D_complexArr (int N)
- int getFilterWidth (Database *db)
- int getModelSize (Database *db)
- int getDNSModelSize (Database *db)
- void createLES_DB (std::string source_dir, filter_base &f, std::string out_dir)
- void createDNS_DB (std::string source_dir, std::string out_dir)
- FilteredData getFilteredVelosity (point &p, Database *db)
- void getFilteredVelosity (LIST_POINTS &pl, LIST_DATA &ld, Database *db)
- void getDNSVelosity (point &p, double &u, double &v, double &w, Database *db)
- void spectr (size &sz, int z, double *in, FILE *fp)
- string getDbType (Database *db)
- size getDomainSize (Database *db)

8.1.2 Private Member Functions

- int getdir (std::string dir, std::string ext, std::vector< std::string > &vfiles, const std::string &optional="")
- int read_file (string &path, string &fname, size &sz, int iz, double *u, double *v, double *w)
- int indx (size &sz, int x, int y, int z)
- int periodic_indx (int size, int ind)
- double filter (point &, size sz, filter_base &f, double *v)
- void populateDnsDbData (string path, std::string fname, size sz, int z, Database *db)

- void populateDNSDbSource (int, std::string, Database *)
- fftw_plan perform_forward_fft (size &sz, double *in, fftw_complex *out)
- fftw_plan perform_forward_fft (int Nx, double *in, fftw_complex *out)
- fftw_plan perform_forward_fft (int Nx, int Ny, double *in, fftw_complex *out)
- fftw_plan perform_forward_fft (int Nz, int Nx, int Ny, double *in, fftw_complex *out)
- fftw_plan perform_backward_fft (size &sz, fftw_complex *in, double *)

8.1.3 Private Attributes

- plugins::Plugin * m_plugin
- void * m_handle
- int m_model_size
- size m_DNS_size
- size m_size
- int m_fw
- Database * m_db

8.1.4 Constructor & Destructor Documentation

8.1.4.1 core::core ()

```
41          : m_size(0,0,0),m_handle (NULL),m_fw(0),m_db(0)
42  {
43  }
```

8.1.5 Member Function Documentation

8.1.5.1 fftw_complex * core::allocate_1D_complexArr (int *N*)

Referenced by spectr().

```
638   {
639     fftw_complex *arr = new fftw_complex[N];
640     if(arr == NULL) cout << "Can't allocate memory" << endl;
641     return arr;
642   }
```

Here is the caller graph for this function:

8.1.5.2 double * core::alocate_1Darr (int *N*)

Referenced by spectr().

```
630   {
631     double *arr=0;
632     arr = new double[N]();
633     if(arr == NULL) cout << "Can't allocate memory" << endl;
634     return arr;
635   }
```

Here is the caller graph for this function:

8.1.5.3 double * core::alocate_2Darr (int *Nx*, int *Ny*)

Referenced by main().

```
622   {
623     double *arr=0;
624     arr = new double[Nx*Ny]();
625     if(arr == NULL) cout << "Can't allocate memory" << endl;
626     return arr;
627   }
```

Here is the caller graph for this function:

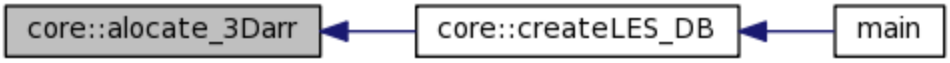
8.1.5.4 double * core::alocate_3Darr (size & *sz*)

References size::lenX, size::lenY, and size::lenZ.

Referenced by createLES_DB().

```
614   {
615       double *arr=0;
616       arr = new double[sz.lenX * sz.lenY * sz.lenZ]();
617       if(arr == NULL) cout << "Can't allocate memory" << endl;
618       return arr;
619   }
```

Here is the caller graph for this function:



8.1.5.5 void core::createDNS_DB (std::string *source_dir*, std::string *out_dir*)

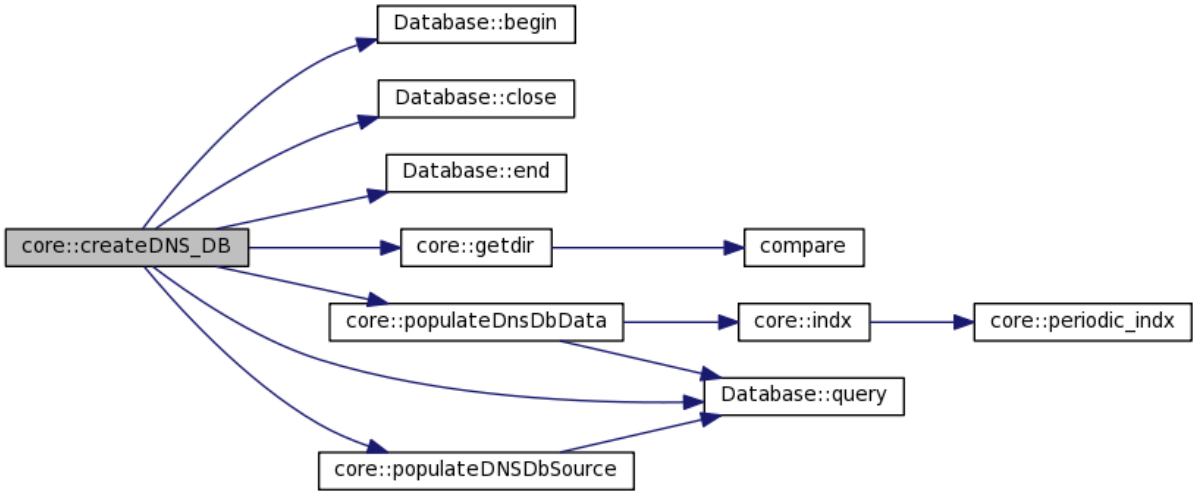
References Database::begin(), Database::close(), Database::end(), getdir(), populateDnsDbData(), populateDNSDbSource(), and Database::query().

Referenced by main().

```
205   {
206     // Out_dir is a absolute path on new database
207     // Check if database already exists..
208     // If database exists - ask override (Y/N)
209    struct stat buffer;
210    if( stat(out_dir.c_str(), &buffer) == 0)
211    {
212     cout << "out_dir " << out_dir << " exist. Override? (Y/N) " <<endl;
213     string choice;
214     getline(cin, choice);
215     while (choice != "n" && choice != "N" && choice != "y" && choice != "Y")
216     {
217       printf ("\nPlease enter Y (Yes) or N (No)\n");
218       getline(cin, choice);
219     }
220       if (choice == "Y" || choice == "y"){ remove(out_dir.c_str());}
221       else return ;
222    }
223
224    Database *db =  new Database(out_dir.c_str());
225
226    db->query((char*)"CREATE TABLE source ( DNS_model_sz INTEGER, path
TEXT);");
227    db->query((char*)"CREATE TABLE data   (ind INTEGER PRIMARY KEY, u
REAL, v REAL , w REAL);");
228    db->query((char*)"CREATE TABLE type   (TYPE TEXT);");
229
230    string stype = "INSERT INTO type VALUES('DNS')";
231    db->query((char*)stype.c_str());
232
233    std::vector<std::string> files;
234    std::string ext="dat";
235    int nf = getdir(source_dir,ext,files);
236    size sz(nf,nf,nf);
237
238    populateDNSDbSource(nf,source_dir,db);
```

```
239
240
241     for(int z = 0; z < files.size();z++)
242     {
243       cout << "reading " << files[z] << endl;
244       db->begin();
245         populateDnsDbData(source_dir,files[z],sz,z,db);
246       db->end();
247     }
248
249
250
251
252     db->close();
253     delete db;
254
255   }
```

Here is the call graph for this function:



Here is the caller graph for this function:

8.1.5.6 void core::createLES_DB (std::string *source_dir*, filter_base & *f*, std::string *out_dir*)

References alocate_3Darr(), Database::begin(), Database::close(), Database::commit(), Database::end(), filter(), filter_base::get_fw(), getdir(), indx(), Database::query(), read_file(), point::x, point::y, and point::z.

Referenced by main().

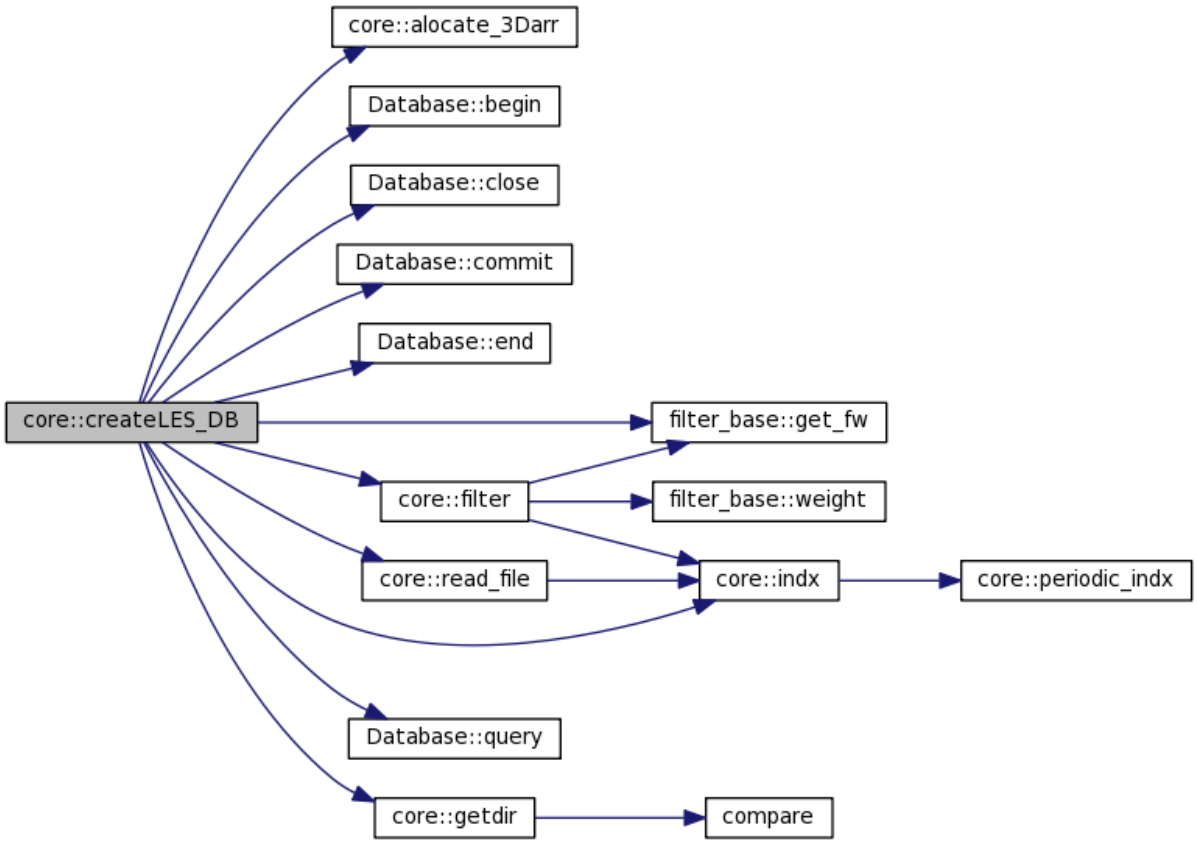
```
391   {
392
393     struct stat buffer;
394     if( stat(out_dir.c_str(), &buffer) == 0)
395     {
396      cout << "out_dir " << out_dir << " exist. Override? (Y/N) " <<endl;
397      string choice;
398      getline(cin, choice);
399      while (choice != "n" && choice != "N" && choice != "y" && choice != "Y")
400      {
401        printf ("\nPlease enter Y (Yes) or N (No)\n");
402        getline(cin, choice);
403      }
404        if (choice == "Y" || choice == "y"){ remove(out_dir.c_str());}
405        else return ;
406     }
407
408
409
410     std::vector<std::string> files;
411     std::string ext="dat";
412     int nf = getdir(source_dir,ext,files);
413     size DNS_sz(nf,nf,nf);
414
415     double* u = alocate_3Darr(DNS_sz);
416     double* v = alocate_3Darr(DNS_sz);
417     double* w = alocate_3Darr(DNS_sz);
418
419
420     for(int z = 0; z < files.size();z++)
421     {
422       cout << "reading " << files[z] << endl;
423       read_file(source_dir,files[z],DNS_sz,z,u,v,w);
424     }
425
426
427     int fw=f.get_fw();
428     int LES_len = nf/fw;
429     int LES_cellNo = 0;
430
431     size LES_model_sz(LES_len,LES_len,LES_len);
432     size LES_cell_sz(fw,fw,fw);
```

```
433
434     char* path = (char*)source_dir.c_str() ;
435
436     Database *db =  new Database(out_dir.c_str());
437
438     db->query((char*)"CREATE TABLE source (filter_width INTEGER, LES_model_sz
INTEGER, path TEXT);");
439     db->query((char*)"CREATE TABLE data   (ind INTEGER PRIMARY KEY,
filtered_u REAL, filtered_v REAL , filtered_w REAL);");
440     db->query((char*)"CREATE TABLE type   (TYPE TEXT);");
441
442     string stype = "INSERT INTO type VALUES('LES')";
443     db->query((char*)stype.c_str());
444
445     stringstream sfw;
446     sfw<<fw;
447     stringstream slen;
448     slen<<LES_len;
449     string ch=",";
450     string en=" );";
451     string qvo="\"";
452
453
454     string insert_source="INSERT INTO source VALUES(";
455     insert_source += sfw.str();
456     insert_source += ch;
457     insert_source += slen.str();
458     insert_source += ch;
459     insert_source += qvo;
460     insert_source += source_dir;
461     insert_source += qvo;
462     insert_source += en;
463
464     //printf("insert_source=[%s\n",insert_source.c_str());
465     db->query((char*)insert_source.c_str());
466     //db->query((char*)"INSERT INTO source VALUES(5,LES_len,path );");
467
468
469   point orig(0,0,0);
470   db->begin();
471     for(int zz=0; zz < LES_len; zz++)
472     {
473       orig.x = 0;
474       for(int xx = 0; xx < LES_len;xx++)
475       {
476         orig.y=0;
```

```
477         for(int yy=0; yy <LES_len;yy++)
478         {
479           int ind = indx(LES_model_sz,xx,yy,zz);
480           double filtred_u = filter(orig,DNS_sz,f,u);
481           double filtred_v = filter(orig,DNS_sz,f,v);
482           double filtred_w = filter(orig,DNS_sz,f,w);
483           LES_cellNo++;
484
485           orig.y += fw;
486           stringstream sind;
487           sind<<ind;
488           stringstream sfiltred_u;
489           sfiltred_u<<filtred_u;
490           stringstream sfiltred_v;
491           sfiltred_v<<filtred_v;
492           stringstream sfiltred_w;
493           sfiltred_w<<filtred_w;
494
495           insert_source="INSERT INTO data VALUES(";
496           insert_source += sind.str();
497           insert_source += ch;
498           insert_source += sfiltred_u.str();
499           insert_source += ch;
500           insert_source += sfiltred_v.str();
501           insert_source += ch;
502           insert_source += sfiltred_w.str();
503           insert_source += en;
504          db->query((char*)insert_source.c_str());
505         }
506         db->commit();
507        orig.x+= fw;
508       }
509      orig.z += fw;
510      printf(" done level %d \n",zz);
511     }
512
513   db->end();
514
515
516
517    delete[] u;
518    delete[] v;
519    delete[] w;
520
521
522    db->close();
```

```
523     delete db;
524
525     cout << out_dir << " Has been successfully created" << endl;
526
527
528   }
```

Here is the call graph for this function:



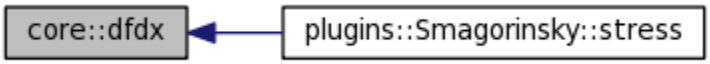
Here is the caller graph for this function:

8.1.5.7 double core::dfdx (int *ind*, order_t *o*, double * *u*)

Referenced by plugins::Smagorinsky::stress().

```
599   {
600     return 0;
601   }
```

Here is the caller graph for this function:



8.1.5.8 double core::dfdy (int *ind*, order_t *o*, double * *u*)

```
604   {
605     return 0;
606   }
```

8.1.5.9 double core::dfdz (int *ind*, order_t *o*, double * *u*)

```
609   {
610     return 0;
611   }
```

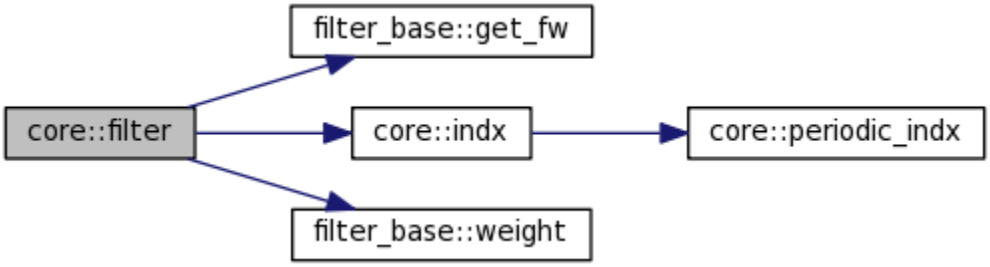
8.1.5.10 double core::filter (point & *pmin*, size *sz*, filter_base & *f*, double * *v*) [private]

References filter_base::get_fw(), indx(), filter_base::weight(), point::x, point::y, and point::z.
Referenced by createLES_DB().

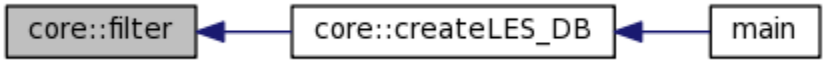
```
531 {
532
533     int fw=f.get_fw();
534     double sum=0;
535
536     point pmax=pmin;
537     pmax.x+=fw;
538     pmax.y+=fw;
539     pmax.z+=fw;
540
541     for(int z =pmin.z; z < pmax.z; z++)
542      for(int x = pmin.x; x < pmax.x; x++)
543       for(int y = pmin.y; y < pmax.y; y++)
544       {
545         int ind = indx(sz,x,y,z); //sz here is DNS model size
546         //position in gaussian weight quibe
547         int xg = x-pmin.x;
548         int yg = y-pmin.y;
549         int zg = z-pmin.z;
550         //printf("Filter ind=%d xg=%d yg=%d zg=%d\n",ind,xg,yg,zg);
```

```
551         sum += v[ind]*f.weight(xg,yg,zg);
552
553       }
554
555     return sum;
556 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



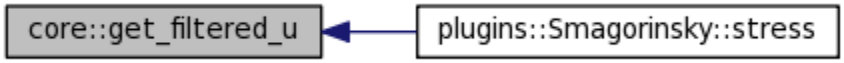
8.1.5.11 int core::get_filter_width ()

8.1.5.12 double * core::get_filtered_u ()

Referenced by plugins::Smagorinsky::stress().

```
583   {
584     return NULL;
585   }
```

Here is the caller graph for this function:



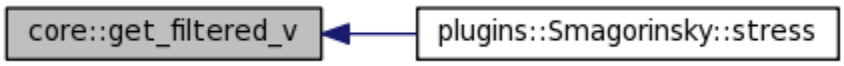
8.1.5.13 double * core::get_filtered_v ()

Referenced by plugins::Smagorinsky::stress().

```
588   {
589     return NULL;
590
```

```
591   }
```

Here is the caller graph for this function:

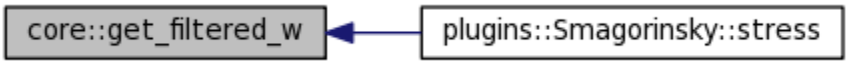


8.1.5.14 double * core::get_filtered_w ()

Referenced by plugins::Smagorinsky::stress().

```
594   {
595     return NULL;
596   }
```

Here is the caller graph for this function:

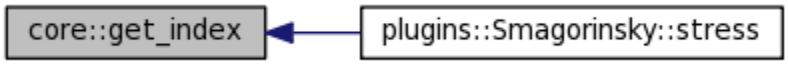


8.1.5.15 int core::get_index (size & *pos*)

Referenced by plugins::Smagorinsky::stress().

```
578   {
579       return 0 ;
580   }
```

Here is the caller graph for this function:



8.1.5.16 int core::get_model_size () [inline]

References m_model_size.

```
20 {return m_model_size;}
```

8.1.5.17 string core::getDbType (Database * *db*)

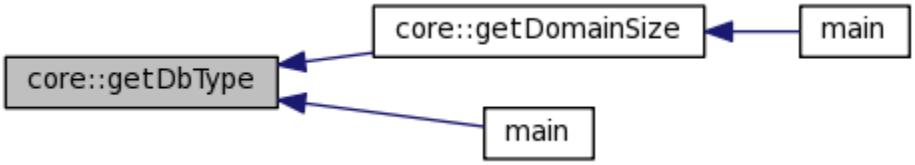
References Database::query().

Referenced by getDomainSize(), and main().

```
104   {
105     table res = db->query((char*)"SELECT TYPE FROM type;");
106     table::iterator it =  res.begin();
107     return ((*it).at(0));
108   }
```

Here is the call graph for this function:

Here is the caller graph for this function:



8.1.5.18 int core::getdir (std::string *dir*, std::string *ext*, std::vector< std::string > & *vfiles*, const std::string & *optional* = "") [private]

References compare().

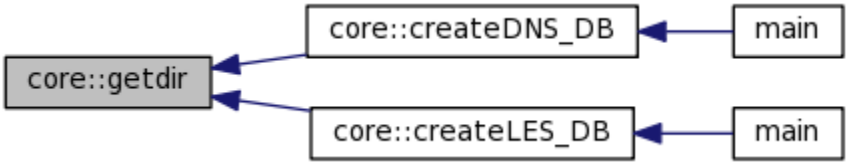
Referenced by createDNS_DB(), and createLES_DB().

```
650   {
651       DIR *dp;
652     struct dirent *dirp;
653     if((dp  = opendir(dir.c_str())) == NULL) {
654         cout << "Error(" << errno << ") opening " << dir << endl;
655         return errno;
656     }
657 
658     list<string> lfiles;
659     const string empty;
660 
661     while ((dirp = readdir(dp)) != NULL) {
662         string file = dirp->d_name;
663 
664      //cout << "IG: file=" <<file << endl;
665      int idx = file.rfind('.');
666         if(idx != string::npos)
667         {
668           if(optional == empty)
669           {
```

```
670             if( file.substr(idx+1) == ext) lfiles.push_back(file);
671           }
672           else
673           {
674             int len = file.length();
675             int opt = optional.length();
676
677
678             if(len > opt)
679             {
680               string substr = file.substr(len-opt,opt);
681               if(substr == optional)  lfiles.push_back(file);
682             }
683
684           }
685         }
686     }
687     lfiles.sort(compare);
688     list<string>::iterator it;
689     for(it=lfiles.begin();it != lfiles.end(); it++)
690     {
691       vfiles.push_back(*it);
692     }
693
694     closedir(dp);
695     return vfiles.size();
696
697   }
```

Here is the call graph for this function:

Here is the caller graph for this function:



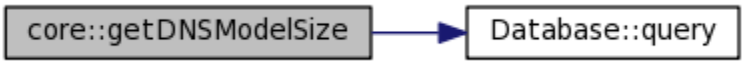
8.1.5.19 int core::getDNSModelSize (Database * *db*)

References Database::query().

Referenced by getDNSVelosity(), and getDomainSize().

```
139   {
140       table res = db->query((char*)"SELECT DNS_model_sz FROM source;");
141       table::iterator it =  res.begin();
142       return atoi((*it).at(0).c_str());
143   }
```

Here is the call graph for this function:



Here is the caller graph for this function:

8.1.5.20 void core::getDNSVelosity (point & *p*, double & *u*, double & *v*, double & *w*, Database * *db*)

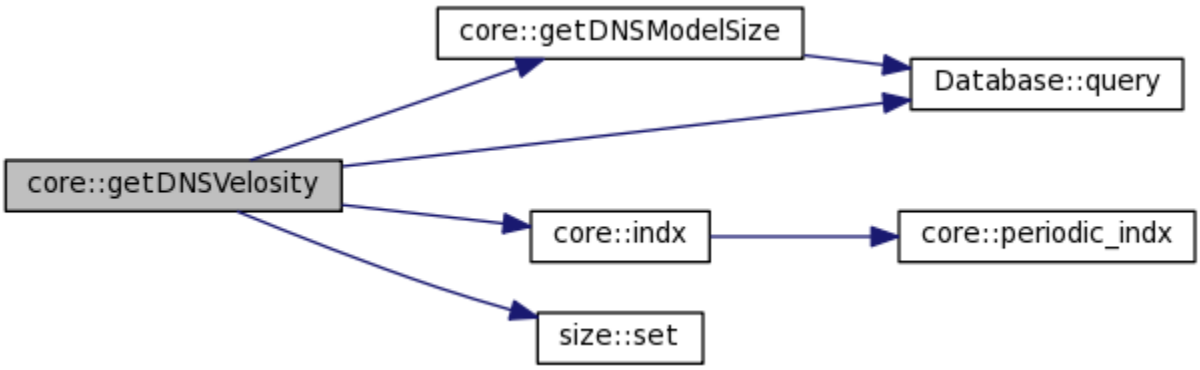
References getDNSModelSize(), indx(), m_db, m_DNS_size, Database::query(), size::set(), point::x, point::y, and point::z.

Referenced by main().

```
348 {
349   if(m_db != db)
350   {
351       m_db = db;
352       int len = getDNSModelSize(db);
353       //cout << " igor - len=" << len << endl;
354       m_DNS_size.set(len,len,len);
355   }
356
357     int ind = indx(m_DNS_size,p.x,p.y,p.z);
358
359     //cout << "igor - ind=" << ind << endl;
360     stringstream sind;
361     sind<<ind;
362     string query="SELECT * FROM data WHERE  ind=";
363     query += sind.str();
364
365     table res = db->query((char*)query.c_str());
```

```
366
367     //cout << " igor - query = " << query << endl;
368
369     table::iterator it;
370     for(it = res.begin(); it < res.end(); ++it)
371     {
372       row  rw = *it;
373       //cout << "ig row = " << rw.at(1).c_str() << endl;
374
375       u  = atof(rw.at(1).c_str());
376       v  = atof(rw.at(2).c_str());
377       w  = atof(rw.at(3).c_str());
378
379       //cout << "u= " << u << endl;
380     }
381
382   return;
383
384
385 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

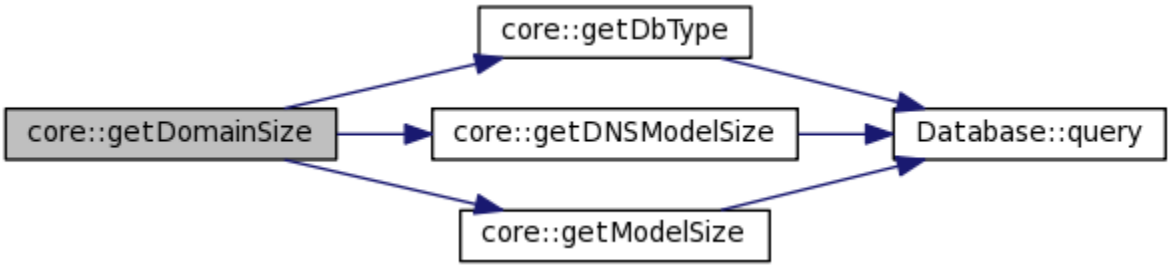
8.1.5.21 size core::getDomainSize (Database * *db*)

References getDbType(), getDNSModelSize(), and getModelSize().
Referenced by main().

```
111   {
```

```
112     int len;
113     if(getDbType(db) == "LES")
114     {
115       len = getModelSize(db);
116     }
117     if(getDbType(db) == "DNS")
118     {
119       len = getDNSModelSize(db);
120     }
121     return size(len,len,len);
122 
123   }
```

Here is the call graph for this function:



Here is the caller graph for this function:

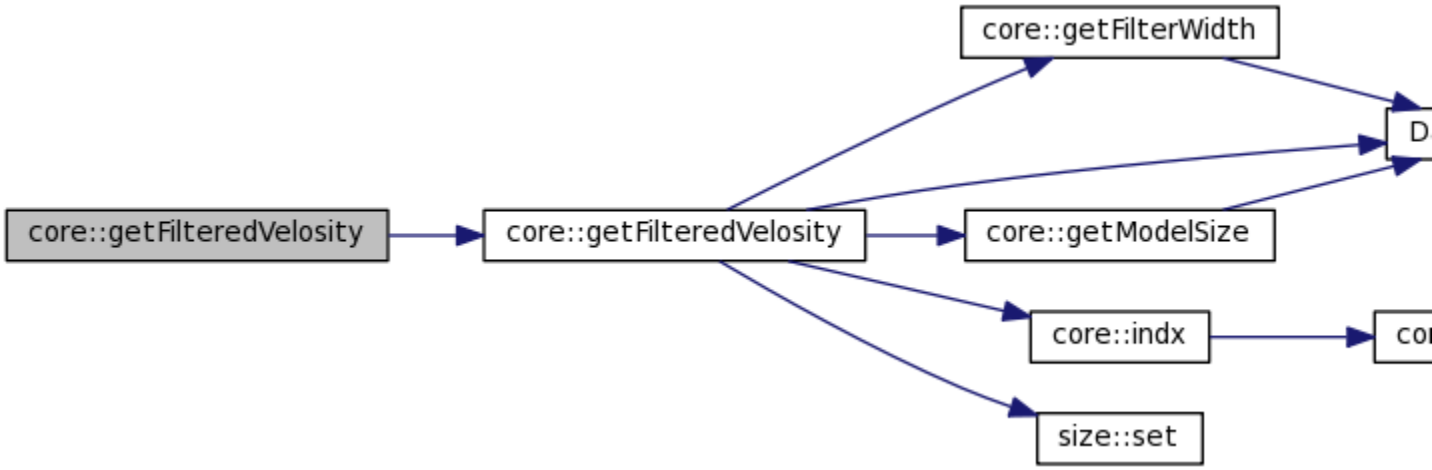
8.1.5.22 void core::getFilteredVelosity (LIST_POINTS & *pl*, LIST_DATA & *ld*, Database * *db*)

References getFilteredVelosity().

```
146   {
147     LIST_POINTS::iterator ip;
148 
149     /*
150     list<int> tl;
151     list<int>::iterator it;
152     tl.push_back(5);
153     for(it = tl.begin() ; it != tl.end(); it++)
154     {
155     }
156     */
157 
```

```
158     for(ip = pl.begin(); ip != pl.end(); ++ip)
159     {
160       point p=*ip;
161       FilteredData fd =  getFilteredVelosity(p,db);
162       dl.push_back(fd);
163     }
164   }
```

Here is the call graph for this function:



8.1.5.23 FilteredData core::getFilteredVelosity (point & *p*, Database * *db*)

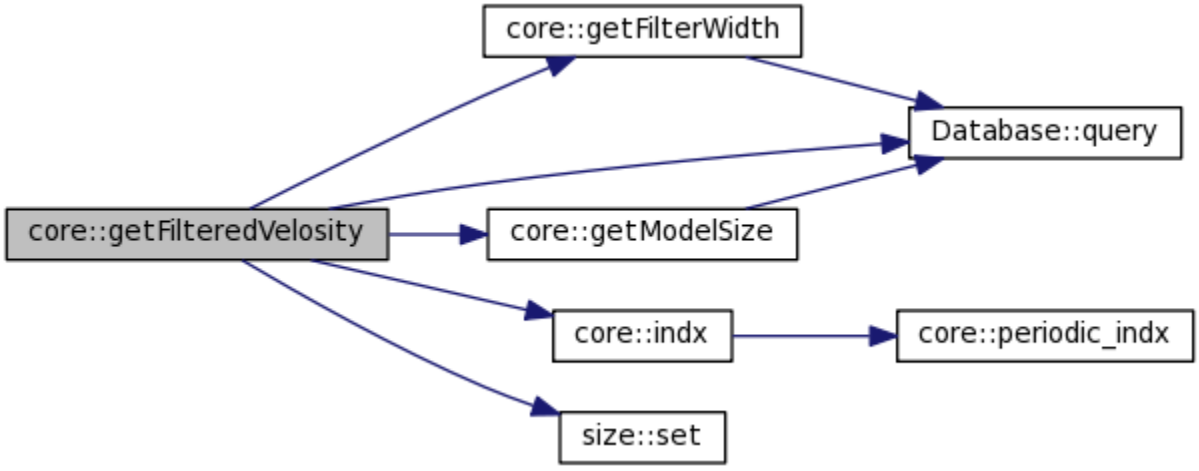
References getFilterWidth(), getModelSize(), indx(), m_db, m_fw, m_size, Database::query(), size::set(), point::x, point::y, and point::z.

Referenced by getFilteredVelosity(), and main().

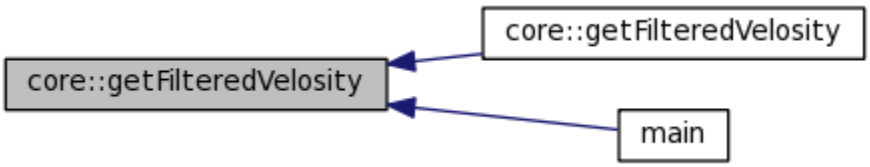
```
167   {
168     if(m_db != db)
169     {
170       m_db = db;
171       m_fw = getFilterWidth(db);
172       int len = getModelSize(db);
173       m_size.set(len,len,len);
174     }
175
176     int ind = indx(m_size,p.x,p.y,p.z);
177     stringstream sind;
178     sind<<ind;
179
180     string query="SELECT * FROM data WHERE  ind=";
181     query += sind.str();
```

```
182
183     table res = db->query((char*)query.c_str());
184     table::iterator it;
185     for(it = res.begin(); it < res.end(); ++it)
186     {
187       row  rw = *it;
188       /*
189       cout << "Values: (ind=" << rw.at(0) <<
190                          ", u=" << rw.at(1) <<
191                          ", v=" << rw.at(2) <<
192                          ", w=" << rw.at(3) <<
193                          ")" << endl;
194        */
195       double u  = atof(rw.at(1).c_str());
196       double v  = atof(rw.at(2).c_str());
197       double w  = atof(rw.at(3).c_str());
198       FilteredData pnt(u,v,w);
199       return pnt;
200
201     }
202   }
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.5.24 int core::getFilterWidth (Database * *db*)

References Database::query().

Referenced by getFilteredVelosity(), and main().

```
126 {
127     table res = db->query((char*)"SELECT filter_width FROM source;");
128     table::iterator it =  res.begin();
129     return atoi((*it).at(0).c_str());
130 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.1.5.25 int core::getModelSize (Database * *db*)

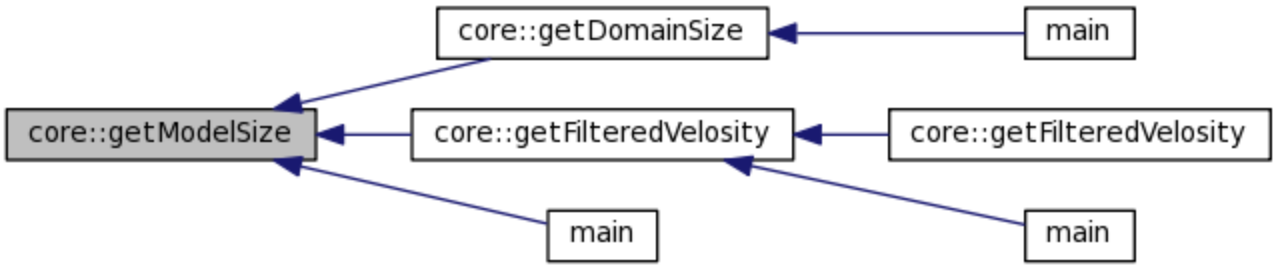
References Database::query().

Referenced by getDomainSize(), getFilteredVelosity(), and main().

```
132 {
133     table res = db->query((char*)"SELECT LES_model_sz FROM source;");
134     table::iterator it =  res.begin();
135     return atoi((*it).at(0).c_str());
136 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

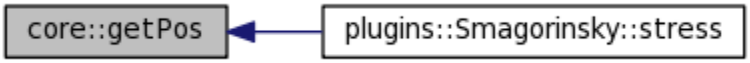


8.1.5.26 size core::getPos ()

Referenced by plugins::Smagorinsky::stress().

```
573   {
574     return size(0,0,0);
575   }
```

Here is the caller graph for this function:



8.1.5.27 int core::indx (size & *sz*, int *x*, int *y*, int *z*) [private]

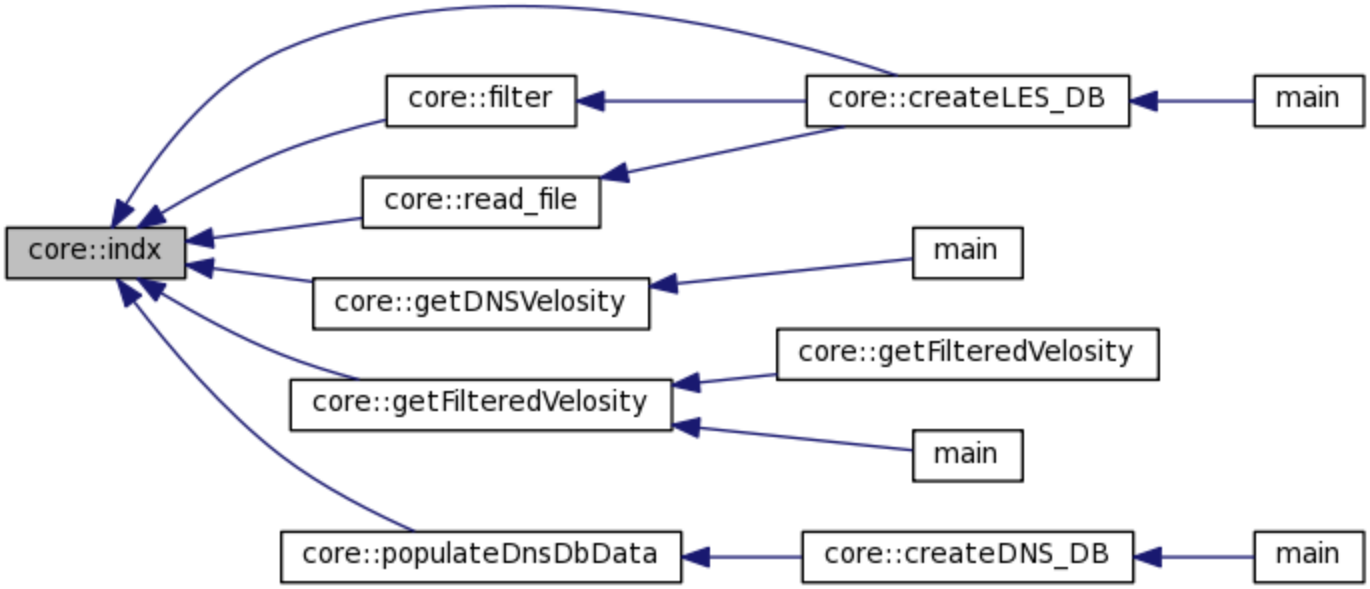
References size::lenX, size::lenY, size::lenZ, and periodic_indx().

Referenced by createLES_DB(), filter(), getDNSVelosity(), getFilteredVelosity(), populateDnsDbData(), and read_file().

```
753 {
754
755   int xp = periodic_indx(sz.lenX,x);
756   int yp = periodic_indx(sz.lenY,y);
757   int zp = periodic_indx(sz.lenZ,z);
758   int ind =  yp + sz.lenY*(xp + sz.lenX*zp);
759   return ind;
760 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



8.1.5.28 void core::load_plugin ()

References construct(), m_handle, and m_plugin.

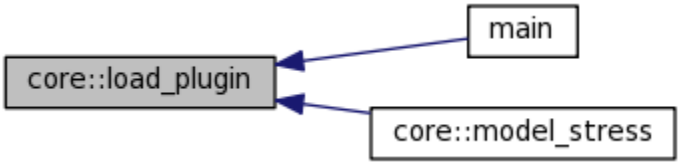
Referenced by main(), and model_stress().

```
46  {
47
48     //void *handle = NULL;
49     if(!(m_handle = dlopen("lib/libplugin.so", RTLD_LAZY)))
50     {
51       std::cerr << "Plugin: " << dlerror() << std::endl;
52       return;
53     }
54     dlerror();
55
56     pluginConstructor construct = (plugins::Plugin* (*)(void)) dlsym(m_handle,
"construct");
57     char *error = NULL;
58     if((error = dlerror()))
59     {
60       std::cerr << "Plugin: " << dlerror() << std::endl;
61       dlclose(m_handle);
62       return;
63     }
64
65     //plugins::Plugin *plugin = construct();
66     //std::cout <<  plugin->toString() << std::endl;
67     //delete plugin;
```

```
68      m_plugin = construct();
69     //dlclose(handle);
70  }
```

Here is the call graph for this function:

Here is the caller graph for this function:

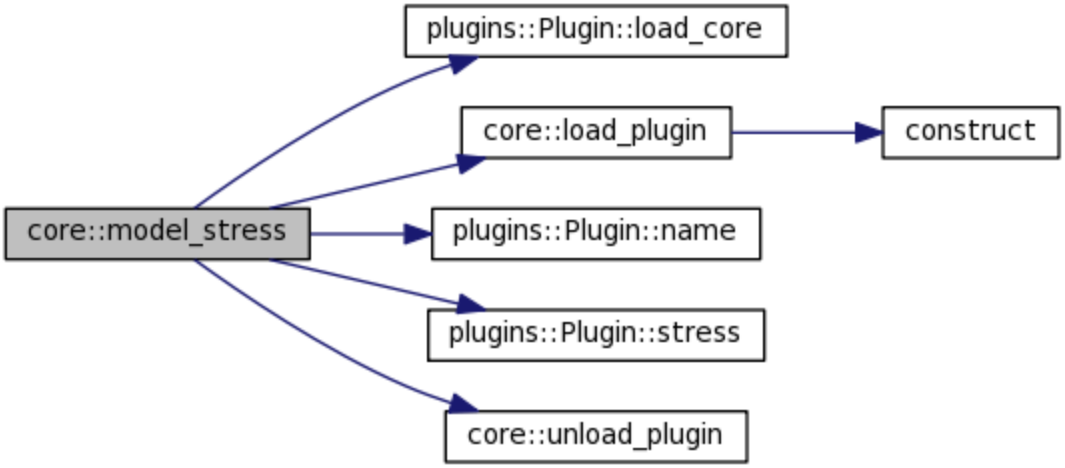


8.1.5.29 void core::model_stress ()

References plugins::Plugin::load_core(), load_plugin(), m_plugin, plugins::Plugin::name(), plugins::Plugin::stress(), and unload_plugin().

```
559   {
560     load_plugin();
561     m_plugin->load_core(this );
562 
563     std::cout << "simulating stress by " << m_plugin->name() << " model" << std::endl;
564 
565     Matrix m(3,3);
566     m_plugin->stress(m);
567 
568     unload_plugin();
569 
570   }
```

Here is the call graph for this function:

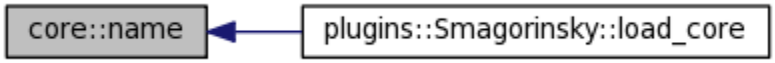


8.1.5.30 std::string core::name ()

Referenced by plugins::Smagorinsky::load_core().

```
86    {
87      return std::string(" The  core");
88    }
```

Here is the caller graph for this function:



8.1.5.31 fftw_plan core::perform_backward_fft (size & *sz*, fftw_complex * *in*, double * *out*) [private]

References size::lenX, size::lenY, and size::lenZ.

```
861 {
862 /*
863  The inverse transforms, taking complex input (storing the non-redundant half of a
logically Hermitian array)
864   to real outputu create a plan and perform  discrete Fourier transform (DFT) in tre
dimensions
865
866  see "http://www.fftw.org/doc/Real_002ddata-DFT-Array-Format.html#Real_002ddata-
DFT-Array-Format"
867  return fftw_plan
868 */
869   int N = sz.lenX*sz.lenY*sz.lenZ;
870   fftw_plan plan_backward;
```

```
871
872   plan_backward = fftw_plan_dft_c2r_3d ( sz.lenZ, sz.lenX, sz.lenY, in, out,
FFTW_ESTIMATE );
873
874   //plan_backward = fftw_plan_dft_c2r_2d ( sz.x, sz.y, in, out, FFTW_ESTIMATE );
875
876    fftw_execute ( plan_backward );
877
878    return plan_backward;
879
880 }
```

8.1.5.32 fftw_plan core::perform_forward_fft (int *Nz*, int *Nx*, int *Ny*, double * *in*, fftw_complex * *out*) [private]

8.1.5.33 fftw_plan core::perform_forward_fft (int *Nx*, int *Ny*, double * *in*, fftw_complex * *out*) [private]

```
782 {
783 /*
784  create a plan and perform  discrete Fourier transform (DFT) in 2D space
785  input -pointer to real array,
786        array should be big enough to hold  Nx*Ny  double values
787  output -pointer to a complex array
788        array will hold Nx*(Ny/2 +1)  complex values
789
790  see "http://www.fftw.org/doc/Real_002ddata-DFT-Array-Format.html#Real_002ddata-
DFT-Array-Format"
791  return fftw_plan
792 */
793   fftw_plan plan_forward = fftw_plan_dft_r2c_2d (  Nx, Ny,  in, out,
FFTW_ESTIMATE );
794   fftw_execute ( plan_forward );
795   fftw_destroy_plan(plan_forward);
796   return plan_forward;
797 }
```

8.1.5.34 fftw_plan core::perform_forward_fft (int *Nx*, double * *in*, fftw_complex * *out*) [private]

```
764 {
765 /*
766  create a plan and perform  discrete Fourier transform (DFT) in 1D space
767  input -pointer to real array,
```

```
768         array should be big enough to hold  Nx  double values
769  output -pointer to a complex array
770         array will hold Nx/2 +1  complex values
771
772  see "http://www.fftw.org/doc/Real_002ddata-DFT-Array-Format.html#Real_002ddata-
DFT-Array-Format"
773  return fftw_plan
774 */
775   fftw_plan plan_forward = fftw_plan_dft_r2c_1d (  Nx,  in, out, FFTW_ESTIMATE );
776   fftw_execute ( plan_forward );
777   fftw_destroy_plan(plan_forward);
778   return plan_forward;
779 }
```

8.1.5.35 fftw_plan core::perform_forward_fft (size & *sz*, double * *in*, fftw_complex * *out*) [private]

References size::lenX, size::lenY, and size::lenZ.

Referenced by spectr().

```
818 {
819 /*
820  create a plan and perform  discrete Fourier transform (DFT) in tre  dimensions
821  input -pointer to real array,
822         array should be big enough to hold sz.LenX*sz.LenY*sz.LenZ double values
823  output -pointer to a complex array
824         array will hold sz.LenZ*sz.LenX*(sz.Leny/2+1) complex values
825
826  see "http://www.fftw.org/doc/Real_002ddata-DFT-Array-Format.html#Real_002ddata-
DFT-Array-Format"
827  return fftw_plan
828 */
829    fftw_plan plan_forward;
830
831    //cout << "in[0]=" << in[0]<< endl;
832    //cout << sz.x << " " << sz.y << endl;
833
834    //cout << "out pntr="<<out<<endl;
835
836     plan_forward = fftw_plan_dft_r2c_3d ( sz.lenZ, sz.lenX, sz.lenY,  in, out,
FFTW_ESTIMATE );
837    //plan_forward = fftw_plan_dft_r2c_2d ( sz.x, sz.y, in, out, FFTW_ESTIMATE );
838    //cout << "about to execute plan" << endl;
839    fftw_execute ( plan_forward );
840    //cout << "about to return" << endl;
841     /*
```

```
842     for ( int i = 0; i < sz.x; i++ )
843     {
844      for ( int j = 0; j < (sz.y/2+1); j++ )
845      {
846
847        int ind = j + sz.y*i;
848        printf ( "  %4d  %4d  %12f  %12f\n",
849           i, j, out[ind][0], out[ind][1] );
850           }
851       }
852
853       cout << "out pntr="<<out<<endl;
854     */
855
856  return plan_forward;
857 }
```

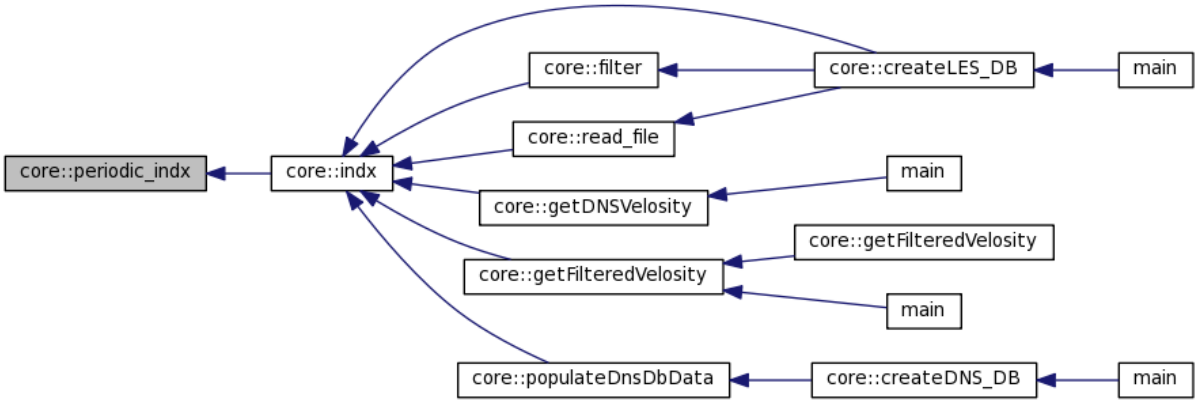
Here is the caller graph for this function:

8.1.5.36 int core::periodic_indx (int *size*, int *ind*) [private]

Referenced by indx().

```
741 {
742   int res;
743   if ( ind < size  &&  ind >= 0) return ind;
744   if ( ind >= size ) { res = ind +1 - size; return res; }
745   if ( ind < 0 )  { res =  ind -1 + size; return res;}
746
747   // should not ever come to this point of return
748   // return unchange indx;
749    return ind;
750 }
```

Here is the caller graph for this function:



8.1.5.37 void core::populateDnsDbData (string *path*, std::string *fname*, size *sz*, int *z*, Database * *db*) [private]

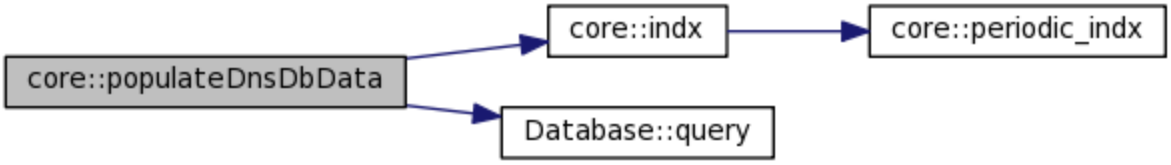
References indx(), size::lenX, and Database::query().

Referenced by createDNS_DB().

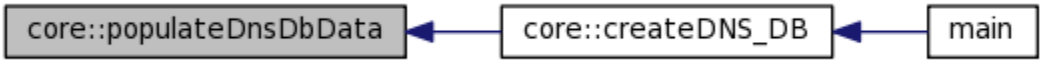
```
282 {
283
284       string fullname=path+fname;
285       ifstream infile(fullname.c_str());
286       string line;
287
288
289       int ix=0;
290       int iy=0;
291       int ind;
292       while (getline(infile, line))
293       {
294         ind = indx(sz,ix,iy,iz);
295
296         istringstream iss(line);
297         double u,v,w;
298         if (!(iss >> u >> v >> w))
299         {
300             break;
301         }
302
303          stringstream sind;
304          sind<<ind;
305
306          stringstream su;
307          su<<u;
308          stringstream sv;
```

```
309         sv<<v;
310         stringstream sw;
311         sw<<w;
312
313         string ch=",";
314         string en=" );";
315         string qvo="\"";
316
317
318         string insert_data="INSERT INTO data VALUES(";
319         insert_data += sind.str();
320         insert_data += ch;
321         insert_data += su.str();
322         insert_data += ch;
323         insert_data += sv.str();
324         insert_data += ch;
325         insert_data += sw.str();
326         insert_data += en;
327
328         //printf("insert_data=%s\n",insert_data.c_str());
329         db->query((char*)insert_data.c_str());
330
331
332
333
334
335       if(ix < sz.lenX) ix++;
336       if(ix == sz.lenX)
337       {
338         ix = 0;
339         iy += 1;
340       }
341
342
343     }
344   infile.close();
345 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.5.38 void core::populateDNSDbSource (int *sz*, std::string *source_dir*, Database * *db*) [private]

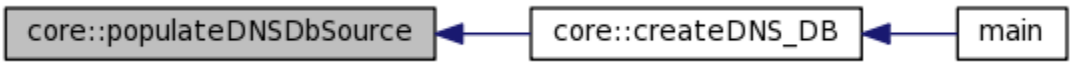
References Database::query().

Referenced by createDNS_DB().

```
258 {
259
260
261     stringstream ssz;
262     ssz<<sz;
263     string ch=",";
264     string en=" );";
265     string qvo="\"";
266
267
268     string insert_source="INSERT INTO source VALUES(";
269     insert_source += ssz.str();
270     insert_source += ch;
271     insert_source += qvo;
272     insert_source += source_dir;
273     insert_source += qvo;
274     insert_source += en;
275
276     //printf("insert_source=[%s\n",insert_source.c_str());
277     db->query((char*)insert_source.c_str());
278
279 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



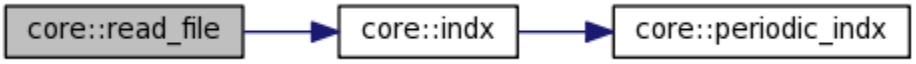
8.1.5.39 int core::read_file (string & *path*, string & *fname*, size & *sz*, int *iz*, double * *u*, double * *v*, double * *w*) [private]

References indx(), and size::lenX.

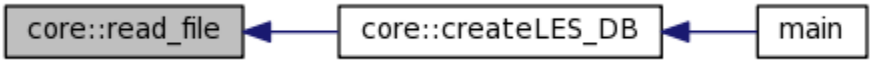
Referenced by createLES_DB().

```
700 {
701       string fullname=path+fname;
702       ifstream infile(fullname.c_str());
703       string line;
704
705
706       int ix=0;
707       int iy=0;
708       int ind;
709       while (getline(infile, line))
710       {
711         //ind = ix + sz.x*(iy + sz.y*iz);
712         //         //ind = iy + sz.y*(ix + sz.x*iz);
713         //
714          ind = indx(sz,ix,iy,iz);
715
716         istringstream iss(line);
717         double v1,v2,v3;
718         if (!(iss >> v1 >> v2 >> v3))
719         {
720             break;
721         }
722
723         u[ind] = v1;
724         v[ind] = v2;
725         w[ind] = v3;
726
727
728         if(ix < sz.lenX) ix++;
729         if(ix == sz.lenX)
730         {
731           ix = 0;
732           iy += 1;
733         }
734
735
736       }
737   infile.close();
738 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.5.40 void core::set_size (size & *s*) [inline]

References m_size.

```
31 {m_size = s;}
```

8.1.5.41 void core::spectr (size & *sz*, int *z*, double * *in*, FILE * *fp*)

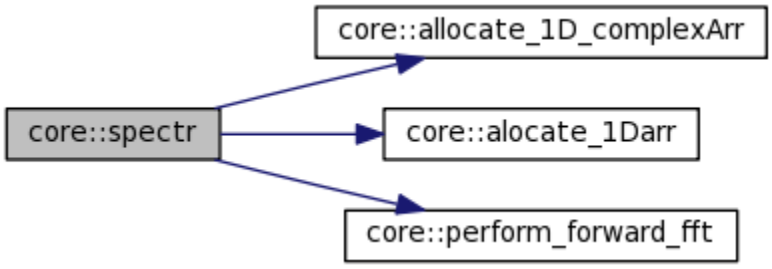
References allocate_1D_complexArr(), alocate_1Darr(), size::lenX, size::lenY, and perform_forward_fft().

Referenced by main().

```
885 {
886   int Ny = sz.lenY;
887   int Nyc = Ny/2+1;
888   int Nx = sz.lenX;
889
890   fftw_complex *out = allocate_1D_complexArr(Nyc);
891   double *sp= alocate_1Darr(Nyc);
892   double *in2= alocate_1Darr(Ny);
893   int ofset = 0;
894
895    for(int yc=0; yc < Nyc; yc++) sp[yc] = 0;
896
897    for(int x = 0; x < Nx; x++)
898    {
899      memcpy(in2,in+ofset,Ny);
900      ofset += Ny;
901      perform_forward_fft(Ny,in2,out);
902      for(int yc=0; yc < Nyc ; yc++)
903      {
904        sp[yc] += out[yc][0]*out[yc][0] +  out[yc][1]*out[yc][1];
905      }
906    }
907
```

```
908    for(int yc=0; yc < Nyc; yc++)
909    {
910        sp[yc] /= Nx;
911        //fprintf(fp,"%12f\n",sp[yc]);
912    }
913
914   //fclose(fp);
915
916   delete[] out;
917    delete[] sp;
918
919 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

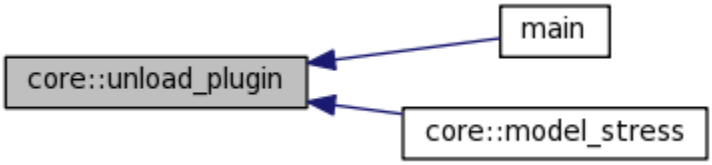
8.1.5.42 void core::unload_plugin ()

References m_handle, and m_plugin.

Referenced by main(), and model_stress().

```
80   {
81      delete m_plugin;
82      dlclose(m_handle);
83   }
```

Here is the caller graph for this function:



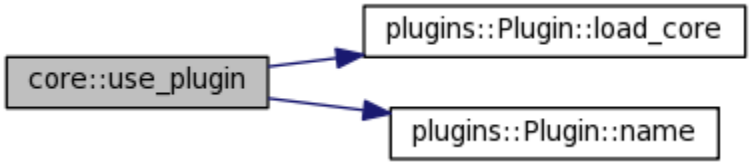
8.1.5.43 void core::use_plugin ()

References plugins::Plugin::load_core(), m_plugin, and plugins::Plugin::name().

Referenced by main().

```
73  {
74    std::cout << "using plugin " <<std::endl;
75    std::cout << m_plugin->name() << std::endl;
76    m_plugin->load_core(this );
77  }
```

Here is the call graph for this function:



Here is the caller graph for this function:

8.1.6 Member Data Documentation

8.1.6.1 Database* core::m_db [private]

Referenced by getDNSVelosity(), and getFilteredVelosity().

8.1.6.2 size core::m_DNS_size [private]

Referenced by getDNSVelosity().

8.1.6.3 int core::m_fw [private]

Referenced by getFilteredVelosity().

8.1.6.4 void* core::m_handle [private]

Referenced by load_plugin(), and unload_plugin().

8.1.6.5 int core::m_model_size [private]

Referenced by get_model_size().

8.1.6.6 plugins::Plugin* core::m_plugin [private]

Referenced by load_plugin(), model_stress(), unload_plugin(), and use_plugin().

8.1.6.7 size core::m_size [private]

Referenced by getFilteredVelosity(), and set_size().

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

- DNS_plugins/include/core.h
- DNS_plugins/src/core.cpp

8.1.6.9

8.2 Database Class Reference

#include <database.h>

8.2.1 Public Member Functions

- Database (const char *filename)
- ~Database ()
- bool open (const char *filename)
- table query (char *query)
- void close ()
- void begin ()
- void commit ()
- void end ()

8.2.2 Private Attributes

- sqlite3 * database

8.2.3 Constructor & Destructor Documentation

8.2.3.1 Database::Database (const char * *filename*)

References database, and open().

```
5 {
6       database = NULL;
7       open(filename);
8 }
```

Here is the call graph for this function:

8.2.3.2 Database::~Database ()

```
11 {
12 }
```

8.2.4 Member Function Documentation

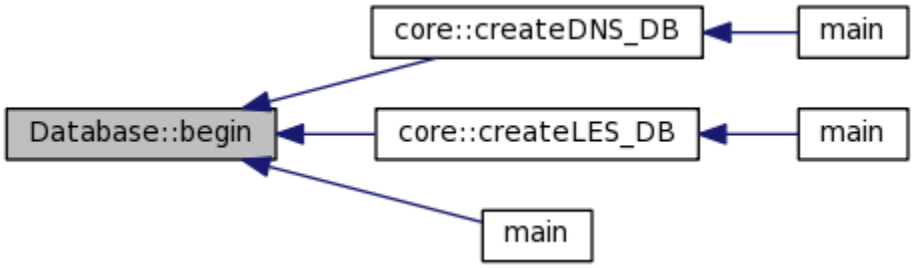
8.2.4.1 void Database::begin ()

References database.

Referenced by core::createDNS_DB(), core::createLES_DB(), and main().

```
23 {
24   sqlite3_exec(database, "BEGIN TRANSACTION;", NULL, NULL, NULL);
25 }
```

Here is the caller graph for this function:



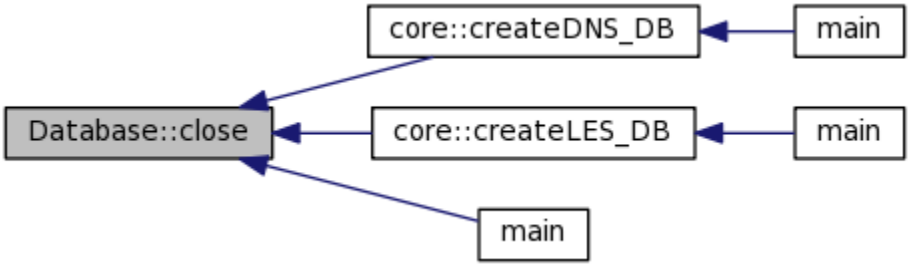
8.2.4.2 void Database::close ()

References database.

Referenced by core::createDNS_DB(), core::createLES_DB(), and main().

```
76 {
77         sqlite3_close(database);
78 }
```

Here is the caller graph for this function:



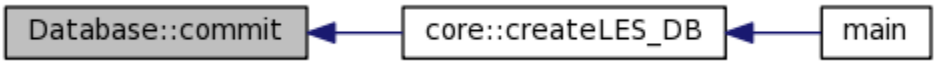
8.2.4.3 void Database::commit ()

References database.

Referenced by core::createLES_DB().

```
32 {
33   sqlite3_exec(database, "COMMIT", NULL, NULL, NULL);
34 }
```

Here is the caller graph for this function:



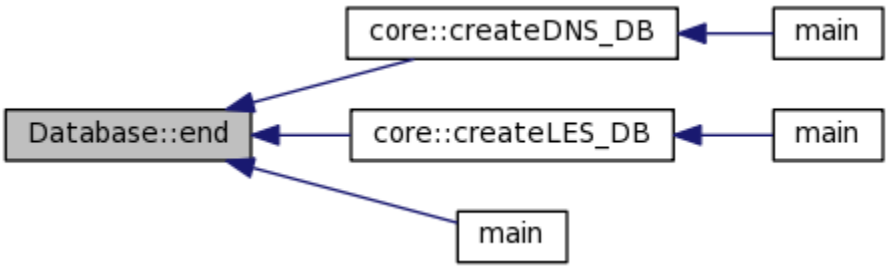
8.2.4.4 void Database::end ()

References database.

Referenced by core::createDNS_DB(), core::createLES_DB(), and main().

```
28 {
29   sqlite3_exec(database, "END TRANSACTION;", NULL, NULL, NULL);
30 }
```

Here is the caller graph for this function:



8.2.4.5 bool Database::open (const char * *filename*)

References database.

Referenced by Database().

```
15 {
16       if(sqlite3_open(filename, &database) == SQLITE_OK)
17             return true;
18
19       return false;
20 }
```

Here is the caller graph for this function:

8.2.4.6 table Database::query (char * *query*)

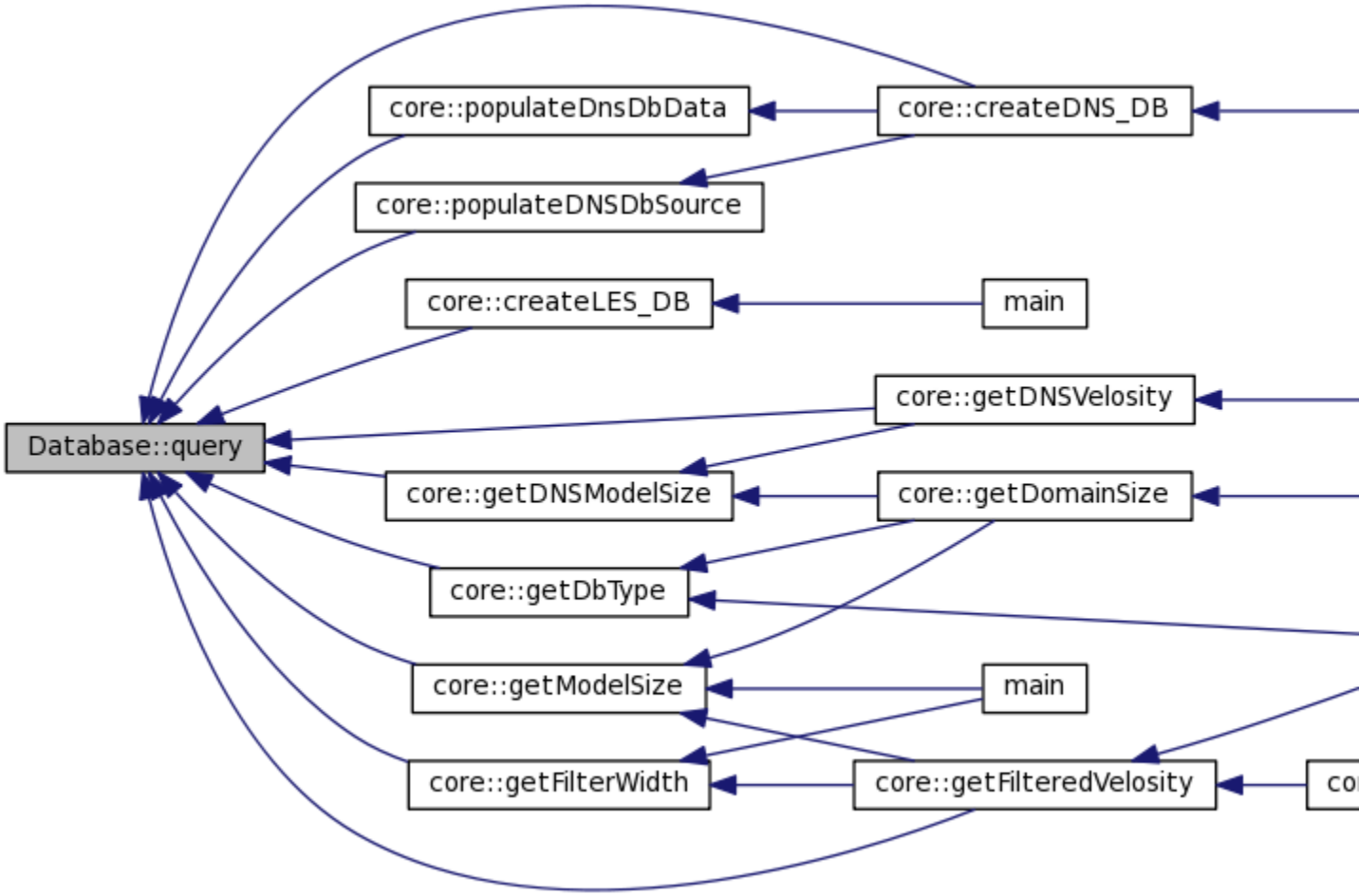
References database.

Referenced by core::createDNS_DB(), core::createLES_DB(), core::getDbType(), core::getDNSModelSize(), core::getDNSVelosity(), core::getFilteredVelosity(), core::getFilterWidth(), core::getModelSize(), core::populateDnsDbData(), and core::populateDNSDbSource().

```
39 {
40     sqlite3_stmt *statement;
41     vector<vector<string> > results;
42
43     if(sqlite3_prepare_v2(database, query, -1, &statement, 0) == SQLITE_OK)
44     {
45         int cols = sqlite3_column_count(statement);
46         int result = 0;
47         while(true)
48         {
49             result = sqlite3_step(statement);
50
51             if(result == SQLITE_ROW)
52             {
53                 vector<string> values;
54                 for(int col = 0; col < cols; col++)
55                 {
56                     values.push_back((char*)sqlite3_column_text(statement, col));
57                 }
58                 results.push_back(values);
59             }
60             else
61             {
62                 break;
63             }
64         }
65
66         sqlite3_finalize(statement);
67     }
68
69     string error = sqlite3_errmsg(database);
70     if(error != "not an error") cout << query << " " << error << endl;
71
72     return results;
```

```
73 }
```

Here is the caller graph for this function:



8.2.5 Member Data Documentation

8.2.5.1 sqlite3* Database::database [private]

Referenced by begin(), close(), commit(), Database(), end(), open(), and query().

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

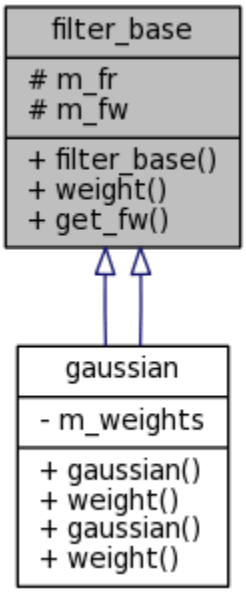
- DNS_plugins/include/database.h
- DNS_plugins/src/database.cpp

8.2.5.3

8.3 filter_base Class Reference

#include <filter.h>

Inheritance diagram for filter_base:



8.3.1 Public Member Functions

- filter_base (int fr)
- virtual double weight (int x, int y, int z)=0
- int get_fw ()

8.3.2 Protected Attributes

- int m_fr
- int m_fw

8.3.3 Constructor & Destructor Documentation

8.3.3.1 filter_base::filter_base (int *fr*) [inline]

References m_fw.

```
7 :m_fr(fr){m_fw = 2*fr+1;};
```

8.3.4 Member Function Documentation

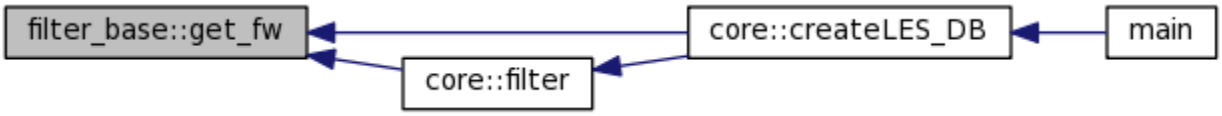
8.3.4.1 int filter_base::get_fw () [inline]

References m_fr.

Referenced by core::createLES_DB(), and core::filter().

```
9 {return 2*m_fr+1;}
```

Here is the caller graph for this function:



8.3.4.2 virtual double filter_base::weight (int *x*, int *y*, int *z*) [pure virtual]

Implemented in gaussian, and gaussian.

Referenced by core::filter().

Here is the caller graph for this function:

8.3.5 Member Data Documentation

8.3.5.1 int filter_base::m_fr [protected]

Referenced by gaussian::gaussian(), and get_fw().

8.3.5.2 int filter_base::m_fw [protected]

Referenced by filter_base(), and gaussian::gaussian().

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

- /home/vu-s3931905/DNS_plugins/include/filter.h

8.3.5.4

8.4 FilteredData Class Reference

#include <point.h>

8.4.1 Public Member Functions

- FilteredData ()
- FilteredData (double u, double v, double w)
- FilteredData & operator= (const FilteredData &pp)
- FilteredData (const FilteredData &pp)

8.4.2 Public Attributes

- POINT3D p

8.4.3 Constructor & Destructor Documentation

8.4.3.1 FilteredData::FilteredData () [inline]

```
11 {};
```

8.4.3.2 FilteredData::FilteredData (double *u*, double *v*, double *w*) [inline]

References p.

```
13  {
14    p[0] = u;
15    p[1] = v;
16    p[2] = w;
17
18  };
```

8.4.3.3 FilteredData::FilteredData (const FilteredData & *pp*) [inline]

References p.

```
29  {
30    p[0] = pp.p[0];
31    p[1] = pp.p[1];
32    p[2] = pp.p[2];
33  };
```

8.4.4 Member Function Documentation

8.4.4.1 FilteredData& FilteredData::operator= (const FilteredData & *pp*) [inline]

References p.

```
21  {
22    p[0] = pp.p[0];
23    p[1] = pp.p[1];
24    p[2] = pp.p[2];
25  };
```

8.4.5 Member Data Documentation

8.4.5.1 POINT3D FilteredData::p

Referenced by FilteredData(), main(), and operator=().

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

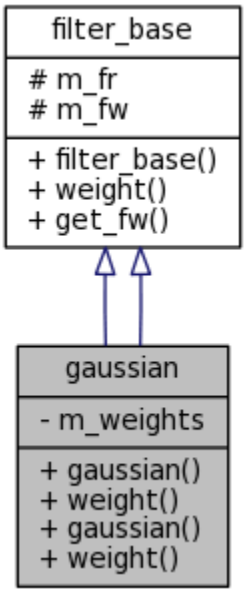
- /home/vu-s3931905/DNS_plugins/include/point.h

8.4.5.3

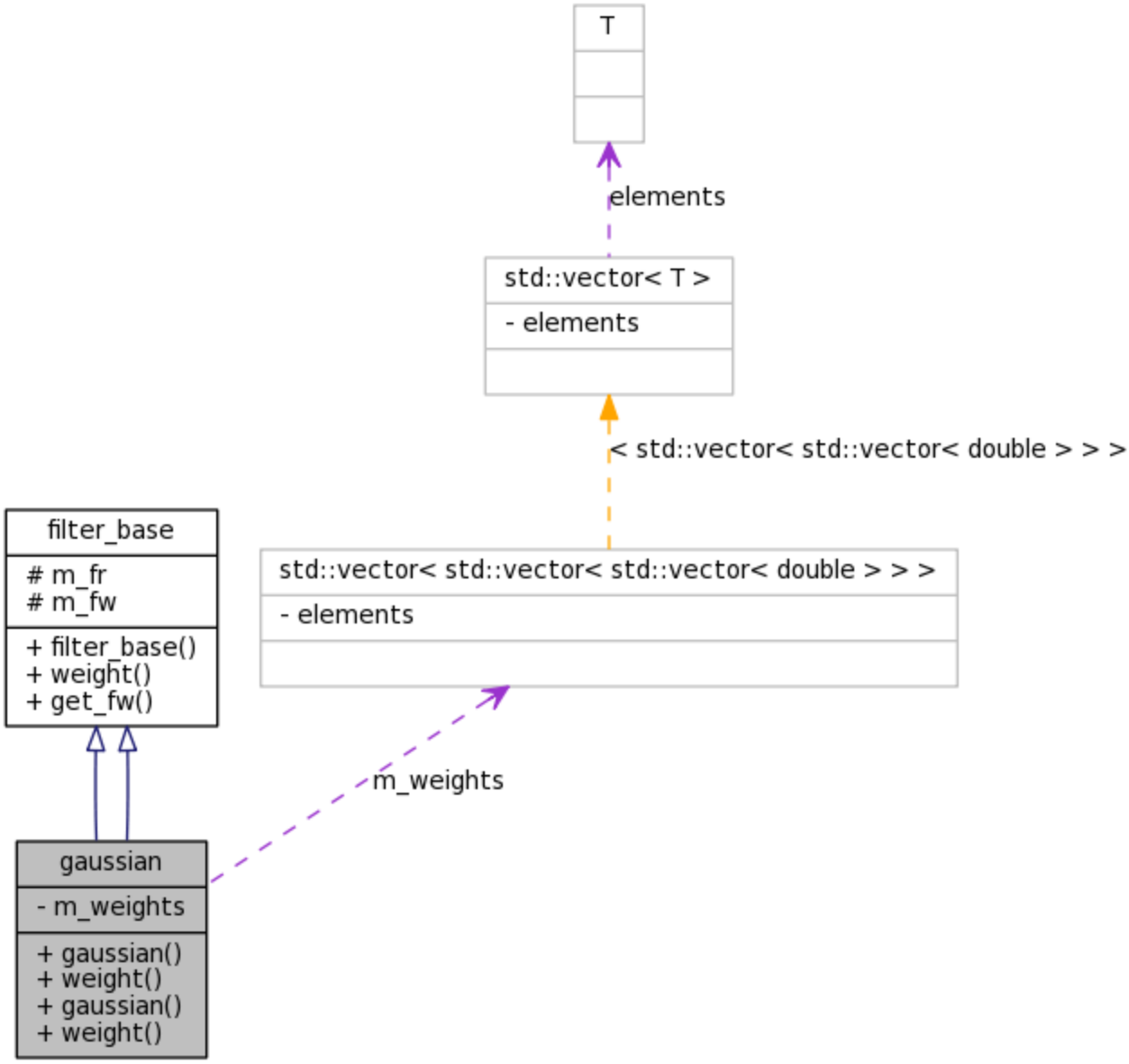
8.5 gaussian Class Reference

#include <gaussian.h>

Inheritance diagram for gaussian:



Collaboration diagram for gaussian:



8.5.1 Public Member Functions

- gaussian (int fr)
- virtual double weight (int x, int y, int z)
- gaussian (int fr)
- virtual double weight (int x, int y, int z)

8.5.2 Private Attributes

- array3D m_weights

8.5.3 Constructor & Destructor Documentation

8.5.3.1 gaussian::gaussian (int *fr*)

References filter_base::m_fr, filter_base::m_fw, and m_weights.

```
6                     :filter_base(fr)
7 {
8   int sp = m_fw*m_fw;
9   printf("m_fw=%d m_fr=%d\n",m_fw,m_fr);
10
11
12   m_weights.resize(m_fw);
13   double sum = 0;
14   for(int z = -m_fr; z <= m_fr; z++)
15   {
16     m_weights[z+m_fr].resize(m_fw);
17     for(int x = -m_fr; x <= m_fr; x++)
18     {
19       m_weights[z+m_fr][x+m_fr].resize(m_fw);
20       for(int y = -m_fr; y <= m_fr; y++)
21       {
22         double w = exp(-6*(x*x + y*y + z*z)/sp);
23         sum += w;
24         m_weights[z+m_fr][x+m_fr][y+m_fr] =  w ;
25       }
26     }
27   }
28
29
30   for(int z = -m_fr; z <= m_fr; z++)
31   {
32     for(int x = -m_fr; x <= m_fr; x++)
33     {
34       for(int y = -m_fr; y <= m_fr; y++)
35       {
36         m_weights[z+m_fr][x+m_fr][y+m_fr] /= sum;
37       }
38     }
39   }
40
41 // check weights
42 /*
43   sum = 0;
44   for(int z = -m_fr; z <= m_fr; z++)
45     for(int x = -m_fr; x <= m_fr; x++)
46       for(int y = -m_fr; y <= m_fr; y++)
```

```
47          sum += m_weights[z+m_fr][x+m_fr][y+m_fr];
48
49 printf("done gaussian sum = %f\n",sum);
50 */
51
52 }
```

8.5.3.2 gaussian::gaussian (int *fr*)

8.5.4 Member Function Documentation

8.5.4.1 virtual double gaussian::weight (int *x*, int *y*, int *z*) [virtual]

Implements filter_base.

8.5.4.2 double gaussian::weight (int *x*, int *y*, int *z*) [virtual]

Implements filter_base.

References m_weights.

```
55 {
56  return m_weights[z][x][y];
57 }
```

8.5.5 Member Data Documentation

8.5.5.1 array3D gaussian::m_weights [private]

Referenced by gaussian(), and weight().

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

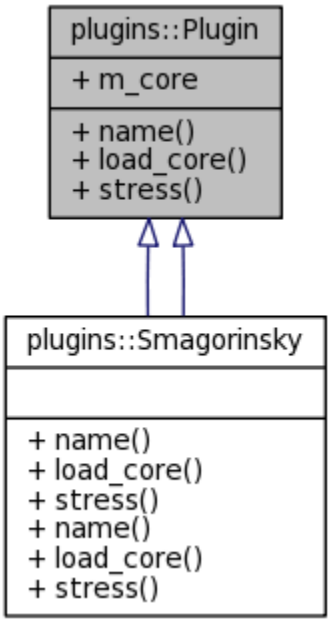
- DNS_plugins/include/gaussian.h
- DNS_plugins/Examples/CreateLESDatabase/include/gaussian.h
- DNS_plugins/src/gaussian.cpp
- DNS_plugins/Examples/CreateLESDatabase/gaussian.cpp

8.5.5.3

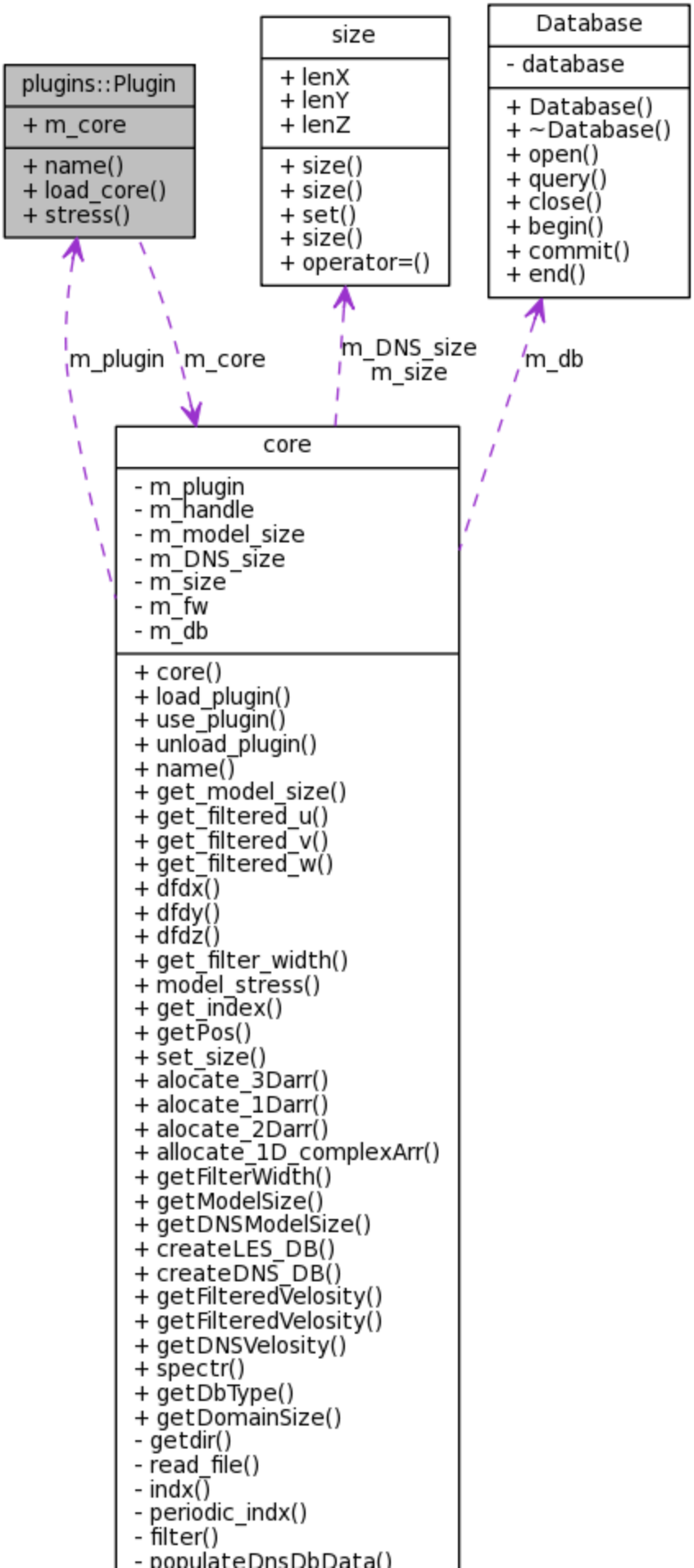
8.6 plugins::Plugin Class Reference

#include <plugin.hpp>

Inheritance diagram for plugins::Plugin:



Collaboration diagram for plugins::Plugin:


plugins::Plugin
+ m_core
+ name()
+ load_core()
+ stress()
size
+ lenX
+ lenY
+ lenZ
+ size()
+ size()
+ set()
+ size()
+ operator=()
Database
- database
+ Database()
+ ~Database()
+ open()
+ query()
+ close()
+ begin()
+ commit()
+ end()
m_plugin
m_core
m_DNS_size
m_size
m_db
core
- m_plugin
- m_handle
- m_model_size
- m_DNS_size
- m_size
- m_fw
- m_db
+ core()
+ load_plugin()
+ use_plugin()
+ unload_plugin()
+ name()
+ get_model_size()
+ get_filtered_u()
+ get_filtered_v()
+ get_filtered_w()
+ dfdx()
+ dfdy()
+ dfdz()
+ get_filter_width()
+ model_stress()
+ get_index()
+ getPos()
+ set_size()
+ alocate_3Darr()
+ alocate_1Darr()
+ alocate_2Darr()
+ allocate_1D_complexArr()
+ getFilterWidth()
+ getModelSize()
+ getDNSModelSize()
+ createLES_DB()
+ createDNS_DB()
+ getFilteredVelosity()
+ getFilteredVelosity()
+ getDNSVelosity()
+ spectr()
+ getDbType()
+ getDomainSize()
- getdir()
- read_file()
- indx()
- periodic_indx()
- filter()
- populateDnsDbData()

8.6.1 Public Member Functions

- virtual std::string name ()=0
- virtual void load_core (core *a)=0
- virtual void stress (Matrix &m)=0

8.6.2 Public Attributes

- core * m_core

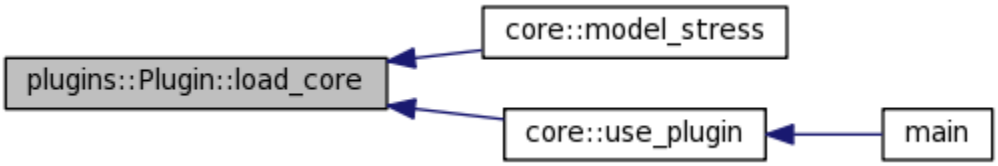
8.6.3 Member Function Documentation

8.6.3.1 virtual void plugins::Plugin::load_core (core * *a*) [pure virtual]

Implemented in plugins::Smagorinsky, and plugins::Smagorinsky.

Referenced by core::model_stress(), and core::use_plugin().

Here is the caller graph for this function:

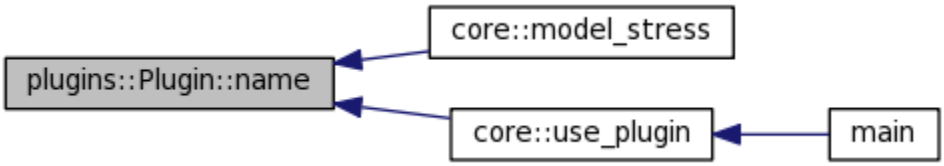


8.6.3.2 virtual std::string plugins::Plugin::name () [pure virtual]

Implemented in plugins::Smagorinsky, and plugins::Smagorinsky.

Referenced by core::model_stress(), and core::use_plugin().

Here is the caller graph for this function:



8.6.3.3 virtual void plugins::Plugin::stress (Matrix & *m*) [pure virtual]

Implemented in plugins::Smagorinsky, and plugins::Smagorinsky.

Referenced by core::model_stress().

Here is the caller graph for this function:

8.6.4 Member Data Documentation

8.6.4.1 core* plugins::Plugin::m_core

Referenced by plugins::Smagorinsky::load_core(), and plugins::Smagorinsky::stress().

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

- /DNS_plugins/include/plugin.hpp

8.6.4.3

8.7 point Class Reference

#include <point.h>

8.7.1 Public Member Functions

- point ()
- point (int init_x, int init_y, int init_z)
- point (const point &s)
- point & operator= (const point &s)
- bool operator< (const point &rhs) const

8.7.2 Public Attributes

- int x
- int y
- int z

8.7.3 Constructor & Destructor Documentation

8.7.3.1 point::point () [inline]

```
41 {};
```

8.7.3.2 point::point (int *init_x*, int *init_y*, int *init_z*) [inline]

References x, y, and z.

```
43   {
44     x = init_x;
45     y = init_y;
46     z = init_z;
47   }
```

8.7.3.3 point::point (const point & *s*) [inline]

```
49 : x(s.x),y(s.y),z(s.z){};
```

8.7.4 Member Function Documentation

8.7.4.1 bool point::operator< (const point & *rhs*) const [inline]

References x, y, and z.

```
62    {
63       if( (x*x + y*y + z*z) < (rhs.x*rhs.x + rhs.y*rhs.y + rhs.z*rhs.z) ) return true;
64       else return false;
65    }
```

8.7.4.2 point& point::operator= (const point & *s*) [inline]

References x, y, and z.

```
52    {
53       x = s.x;
54       y = s.y;
55       z = s.z;
56       return *this;
57    }
```

8.7.5 Member Data Documentation

8.7.5.1 int point::x

Referenced by core::createLES_DB(), core::filter(), core::getDNSVelosity(), core::getFilteredVelosity(), operator<(), operator=(), and point().

8.7.5.2 int point::y

Referenced by core::createLES_DB(), core::filter(), core::getDNSVelosity(), core::getFilteredVelosity(), operator<(), operator=(), and point().

8.7.5.3 int point::z

Referenced by core::createLES_DB(), core::filter(), core::getDNSVelosity(), core::getFilteredVelosity(), operator<(), operator=(), and point().

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

- DNS_plugins/include/point.h

8.7.5.5

8.8 size Class Reference

#include <plugin.hpp>

8.8.1 Public Member Functions

- size ()
- size (int init_x, int init_y, int init_z)
- void set (int init_x, int init_y, int init_z)
- size (const size &s)
- size & operator= (const size &s)

8.8.2 Public Attributes

- int lenX
- int lenY
- int lenZ

8.8.3 Constructor & Destructor Documentation

8.8.3.1 size::size () [inline]

```
20 {};
```

8.8.3.2 size::size (int *init_x*, int *init_y*, int *init_z*) [inline]

References lenX, lenY, and lenZ.

```
22      {
23          lenX = init_x;
24          lenY = init_y;
25          lenZ = init_z;
26      }
```

8.8.3.3 size::size (const size & *s*) [inline]

```
34 : lenX(s.lenX),lenY(s.lenY),lenZ(s.lenZ){};
```

8.8.4 Member Function Documentation

8.8.4.1 size& size::operator= (const size & *s*) [inline]

References lenX, lenY, and lenZ.

```
37      {
38              lenX = s.lenX;
39              lenY = s.lenY;
40              lenZ = s.lenZ;
41              return *this;
42  }
```

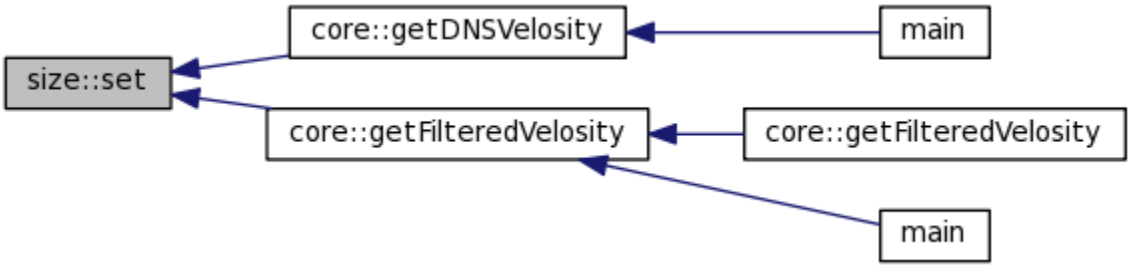
8.8.4.2 void size::set (int *init_x*, int *init_y*, int *init_z*) [inline]

References lenX, lenY, and lenZ.

Referenced by core::getDNSVelosity(), and core::getFilteredVelosity().

```
28      {
29              lenX = init_x;
30              lenY = init_y;
31              lenZ = init_z;
32      }
```

Here is the caller graph for this function:



8.8.5 Member Data Documentation

8.8.5.1 int size::lenX

Referenced by core::alocate_3Darr(), core::indx(), main(), operator=(), core::perform_backward_fft(), core::perform_forward_fft(), core::populateDnsDbData(), core::read_file(), set(), size(), and core::spectr().

8.8.5.2 int size::lenY

Referenced by core::alocate_3Darr(), core::indx(), operator=(), core::perform_backward_fft(), core::perform_forward_fft(), set(), size(), and core::spectr().

8.8.5.3 int size::lenZ

Referenced by core::alocate_3Darr(), core::indx(), operator=(), core::perform_backward_fft(), core::perform_forward_fft(), set(), and size().

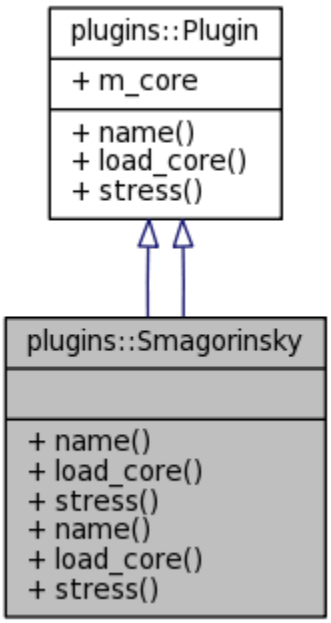
8.8.5.4 The documentation for this class was generated from the following file:

- /home/vu-s3931905/DNS_plugins/include/plugin.hpp

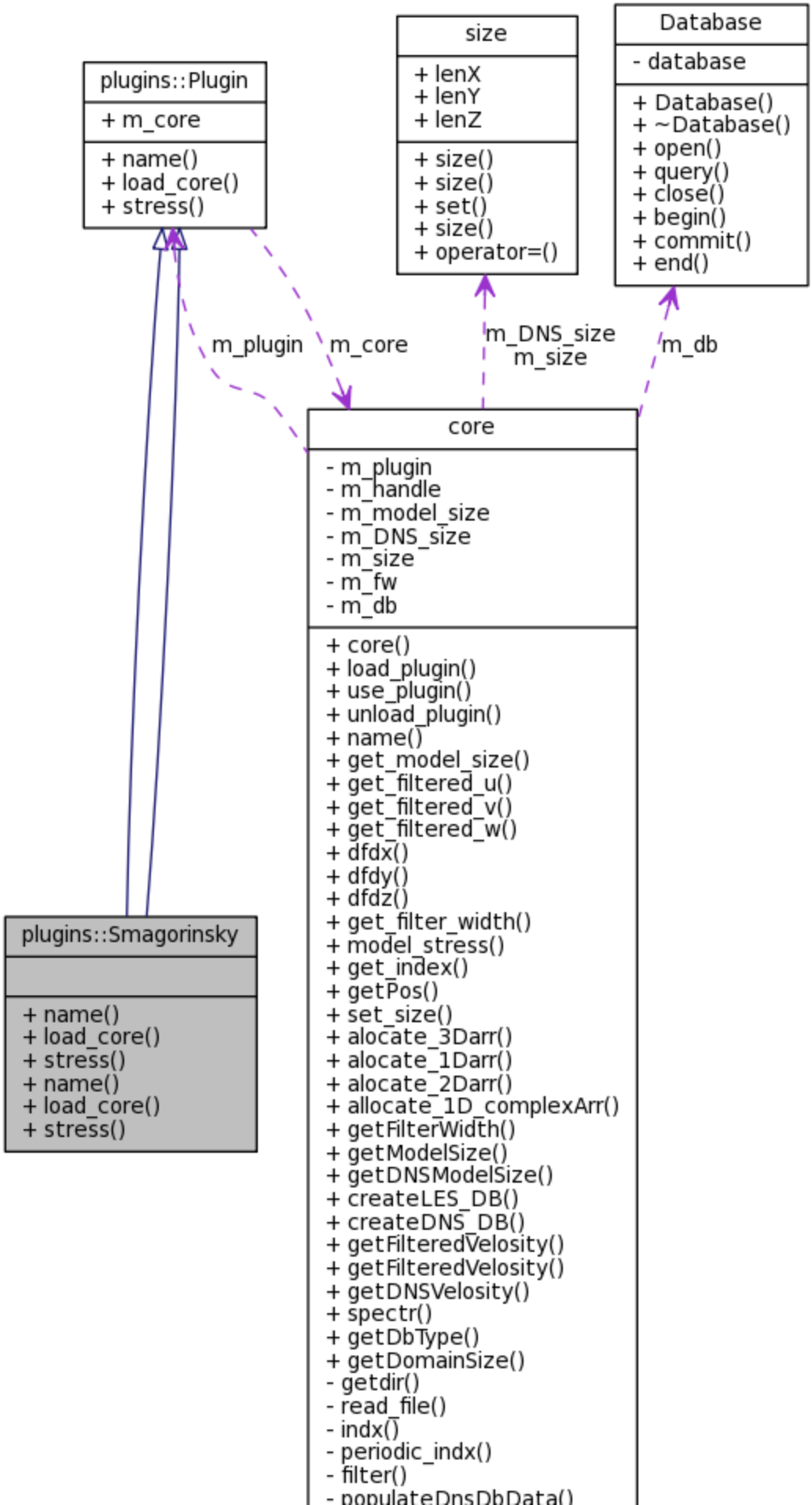
8.8.5.5

8.9 plugins::Smagorinsky Class Reference

Inheritance diagram for plugins::Smagorinsky:



Collaboration diagram for plugins::Smagorinsky:


plugins::Plugin
+ m_core
+ name()
+ load_core()
+ stress()
size
+ lenX
+ lenY
+ lenZ
+ size()
+ size()
+ set()
+ size()
+ operator=()
Database
- database
+ Database()
+ ~Database()
+ open()
+ query()
+ close()
+ begin()
+ commit()
+ end()
m_plugin
m_core
m_DNS_size
m_size
m_db
core
- m_plugin
- m_handle
- m_model_size
- m_DNS_size
- m_size
- m_fw
- m_db
+ core()
+ load_plugin()
+ use_plugin()
+ unload_plugin()
+ name()
+ get_model_size()
+ get_filtered_u()
+ get_filtered_v()
+ get_filtered_w()
+ dfdx()
+ dfdy()
+ dfdz()
+ get_filter_width()
+ model_stress()
+ get_index()
+ getPos()
+ set_size()
+ alocate_3Darr()
+ alocate_1Darr()
+ alocate_2Darr()
+ allocate_1D_complexArr()
+ getFilterWidth()
+ getModelSize()
+ getDNSModelSize()
+ createLES_DB()
+ createDNS_DB()
+ getFilteredVelosity()
+ getFilteredVelosity()
+ getDNSVelosity()
+ spectr()
+ getDbType()
+ getDomainSize()
- getdir()
- read_file()
- indx()
- periodic_indx()
- filter()
- populateDnsDbData()
plugins::Smagorinsky
+ name()
+ load_core()
+ stress()
+ name()
+ load_core()
+ stress()

8.9.1 Public Member Functions

- std::string name ()
- void load_core (core *a)
- void stress (Matrix &s)
- std::string name ()
- void load_core (core *a)
- void stress (Matrix &s)

8.9.2 Member Function Documentation

8.9.2.1 void plugins::Smagorinsky::load_core (core * *a*) [inline, virtual]

Implements plugins::Plugin.

References plugins::Plugin::m_core, and core::name().

```
19   {
20     std::cout << a->name() << std::endl;
21     m_core = a;
22   }
```

Here is the call graph for this function:

8.9.2.2 void plugins::Smagorinsky::load_core (core * *a*) [inline, virtual]

Implements plugins::Plugin.

References plugins::Plugin::m_core, and core::name().

```
19   {
20     std::cout << a->name() << std::endl;
21     m_core = a;
22   }
```

Here is the call graph for this function:

8.9.2.3 std::string plugins::Smagorinsky::name () [inline, virtual]

Implements plugins::Plugin.

```
15    {
16      return std::string("Smagorinsky model plugin");
17    }
```

8.9.2.4 std::string plugins::Smagorinsky::name () [inline, virtual]

Implements plugins::Plugin.

```
15    {
16      return std::string("Graham and Igor Smagorinsky model plugin");
17    }
```

8.9.2.5 void plugins::Smagorinsky::stress (Matrix & *s*) [inline, virtual]

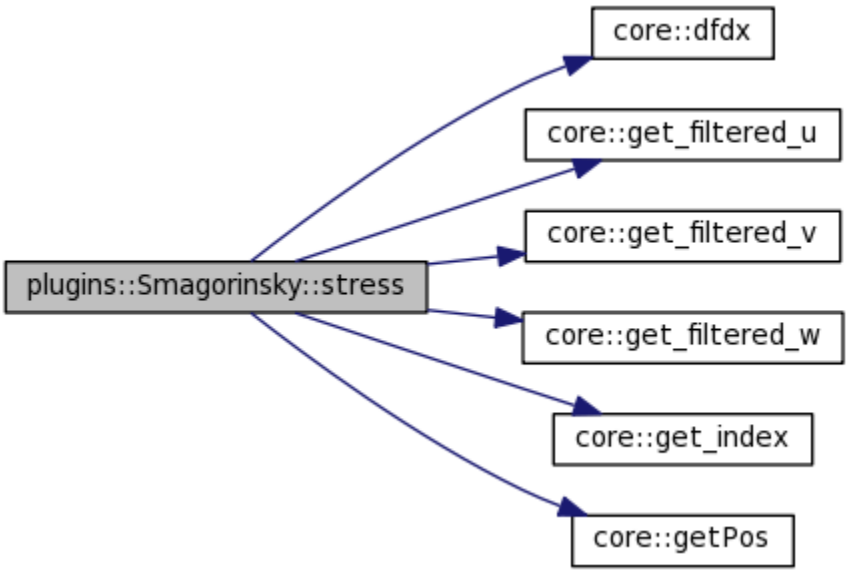
Implements plugins::Plugin.

References core::dfdx(), FOUR, core::get_filtered_u(), core::get_filtered_v(), core::get_filtered_w(), core::get_index(), core::getPos(), and plugins::Plugin::m_core.

```
24    {
25      double *u = m_core->get_filtered_u();
26      double *v = m_core->get_filtered_v();
27      double *w = m_core->get_filtered_w();
28      size pos = m_core->getPos();
29      int ind = m_core->get_index(pos);
30
31      order_t order=FOUR;
32
33 /*
34      double dudx = m_core->dfdx(ind,order,u);
35      double dudy = m_core->dfdy(ind,order,u);
36      double dudz = m_core->dfdz(ind,order,u);
37
38
39      double dvdx = m_core->dfdx(ind,order,v);
40      double dvdy = m_core->dfdy(ind,order,v);
41      double dvdz = m_core->dfdz(ind,order,v);
42
43      double dwdx = m_core->dfdx(ind,order,w);
44      double dwdy = m_core->dfdy(ind,order,w);
45      double dwdz = m_core->dfdz(ind,order,w);
46 */
47      double dudx = m_core->dfdx(ind,order,u);
```

```
48      double dudy = 0;
49      double dudz = 0;
50
51
52      double dvdx = 0;
53      double dvdy = 0;
54      double dvdz = 0;
55
56      double dwdx = 0;
57      double dwdy = 0;
58      double dwdz = 0;
59
60
61
62
63      s(0,0) = -0.5*( dudx + dudx );
64      s(0,1) = -0.5*( dudy + dvdx );
65      s(0,2) = -0.5*( dudz + dwdx );
66      s(1,1) = -0.5*( dvdy + dvdy );
67      s(1,2) = -0.5*( dvdz + dwdx );
68      s(2,2) = -0.5*( dwdz + dwdz );
69
70      s(1,0) = s(0,1);
71      s(2,0) = s(0,2);
72      s(2,1) = s(1,2);
73
74
75    }
```

Here is the call graph for this function:



8.9.2.6 void plugins::Smagorinsky::stress (Matrix & *s*) [inline, virtual]

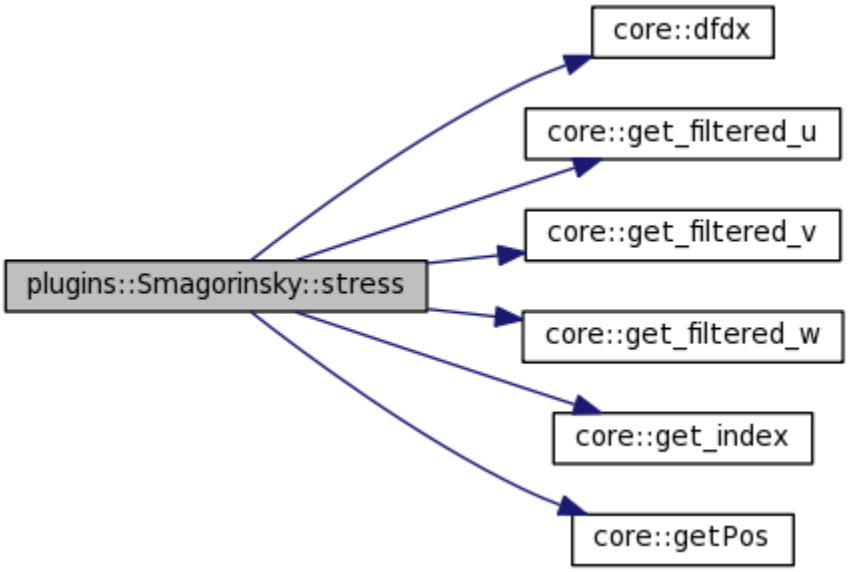
Implements plugins::Plugin.

References core::dfdx(), FOUR, core::get_filtered_u(), core::get_filtered_v(), core::get_filtered_w(), core::get_index(), core::getPos(), and plugins::Plugin::m_core.

```
24   {
25     double *u = m_core->get_filtered_u();
26     double *v = m_core->get_filtered_v();
27     double *w = m_core->get_filtered_w();
28     size pos = m_core->getPos();
29     int ind = m_core->get_index(pos);
30 
31     order_t order=FOUR;
32 
33 /*
34     double dudx = m_core->dfdx(ind,order,u);
35     double dudy = m_core->dfdy(ind,order,u);
36     double dudz = m_core->dfdz(ind,order,u);
37 
38 
39     double dvdx = m_core->dfdx(ind,order,v);
40     double dvdy = m_core->dfdy(ind,order,v);
41     double dvdz = m_core->dfdz(ind,order,v);
42 
43     double dwdx = m_core->dfdx(ind,order,w);
44     double dwdy = m_core->dfdy(ind,order,w);
45     double dwdz = m_core->dfdz(ind,order,w);
46 */
47     double dudx = m_core->dfdx(ind,order,u);
48     double dudy = 0;
49     double dudz = 0;
50 
51 
52     double dvdx = 0;
53     double dvdy = 0;
54     double dvdz = 0;
55 
56     double dwdx = 0;
57     double dwdy = 0;
58     double dwdz = 0;
59 
60 
61 
62 
```

```
63     s(0,0) = -0.5*( dudx + dudx );
64     s(0,1) = -0.5*( dudy + dvdx );
65     s(0,2) = -0.5*( dudz + dwdx );
66     s(1,1) = -0.5*( dvdy + dvdy );
67     s(1,2) = -0.5*( dvdz + dwdx );
68     s(2,2) = -0.5*( dwdz + dwdz );
69
70     s(1,0) = s(0,1);
71     s(2,0) = s(0,2);
72     s(2,1) = s(1,2);
73
74
75   }
```

Here is the call graph for this function:



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

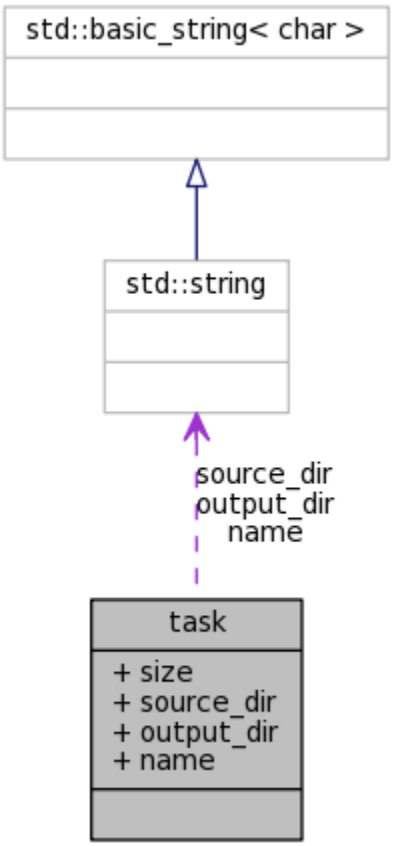
- DNS_plugins/src/plugin.cpp
- DNS_plugins/Examples/Plugins/plugin.cpp

8.9.2.8

8.10 task Struct Reference

#include <task.h>

Collaboration diagram for task:



8.10.1 Public Attributes

- int size
- string source_dir
- string output_dir
- string name

8.10.2 Member Data Documentation

8.10.2.1 string task::name

8.10.2.2 string task::output_dir

8.10.2.3 int task::size

8.10.2.4 string task::source_dir

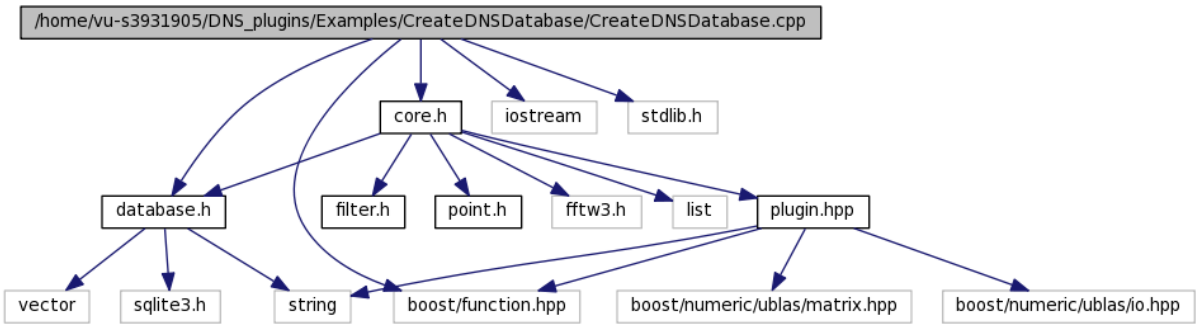
8.10.2.5 The documentation for this struct was generated from the following file:

- DNS_plugins/include/task.h

9 File Documentation

DNS_plugins/Examples/CreateDNSDatabase/CreateDNSDatabase.cpp File Reference

#include "core.h"
#include "database.h"
#include <boost/function.hpp>
#include <iostream>
#include <stdlib.h>
Include dependency graph for CreateDNSDatabase.cpp:



9.1.1 Functions

- int main (int argc, char **argv)

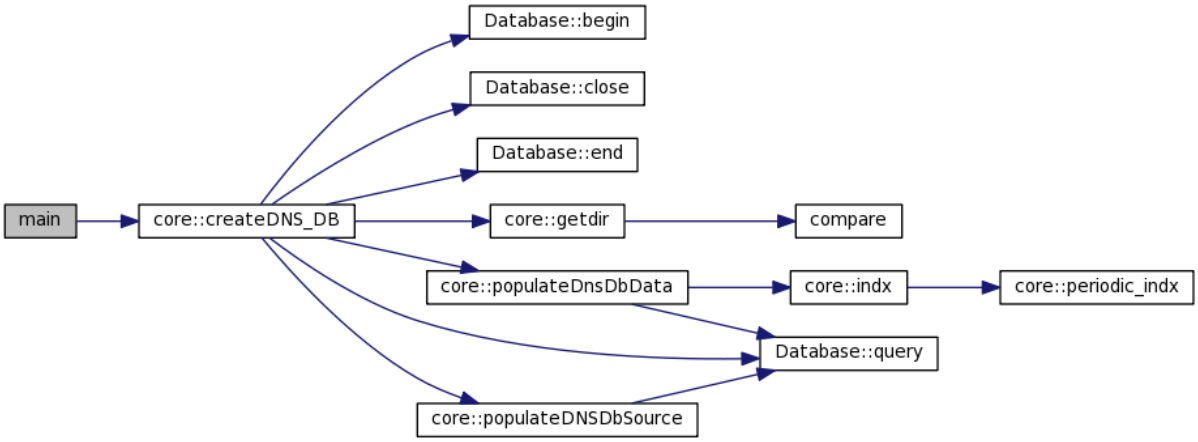
9.1.2 Function Documentation

9.1.2.1 int main (int *argc*, char ** *argv*)

References core::createDNS_DB().

```
12 {
13
14 core *a = new(core);
15
16 string source_dir = string("/home/projects/pVict0004/512/DnsData/");
17 string out_dir = "/home/projects/pVict0004/512/DnsDataDB/DNS_DB";
18 a->createDNS_DB(source_dir, out_dir);
19
20
21
22 delete a;
23 }
```

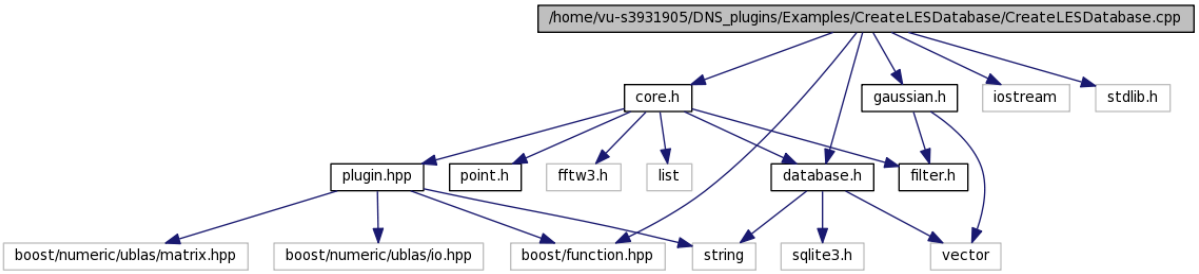
Here is the call graph for this function:


Database::begin
Database::close
Database::end
main
core::createDNS_DB
core::getdir
compare
core::populateDnsDbData
core::indx
core::periodic_indx
Database::query
core::populateDNSDbSource

DNS_plugins/Examples/CreateLESDatabase/CreateLESDatabase.cpp File Reference

```
#include "core.h"
#include "database.h"
#include <boost/function.hpp>
#include "gaussian.h"
#include <iostream>
#include <stdlib.h>
```

Include dependency graph for CreateLESDatabase.cpp:



9.1.3 Functions

- int main (int argc, char **argv)

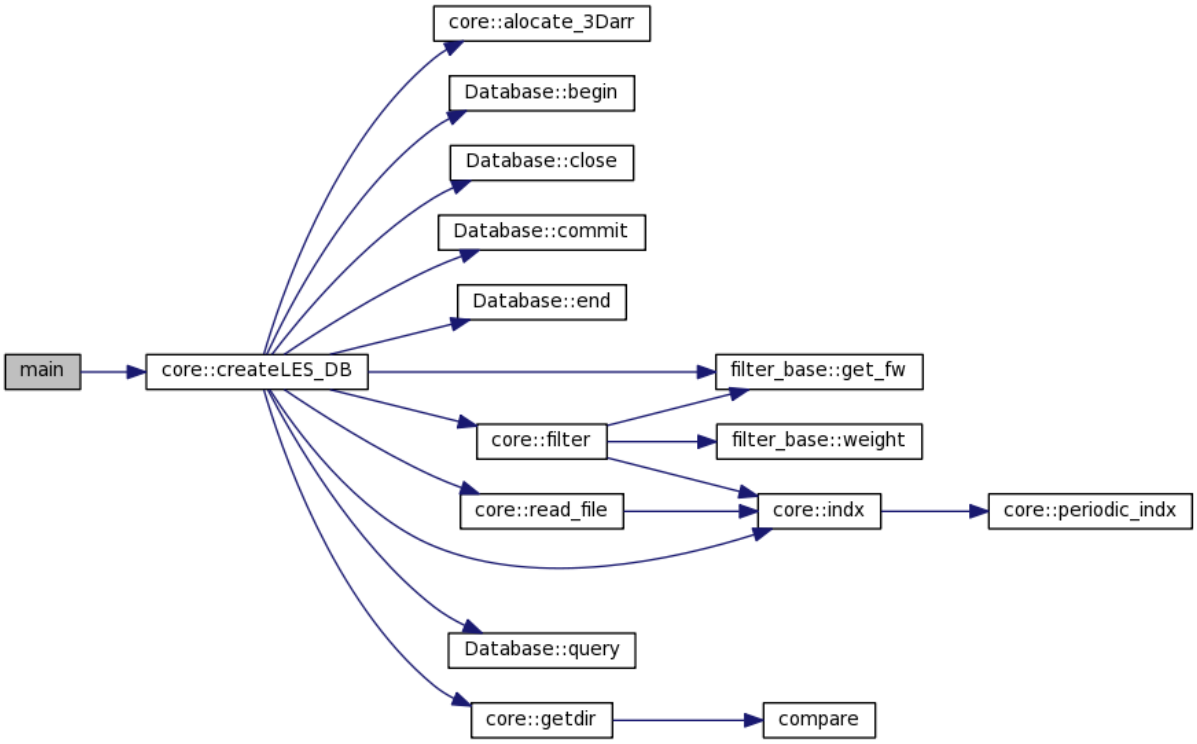
9.1.4 Function Documentation

9.1.4.1 int main (int *argc*, char ** *argv*)

References core::createLES_DB().

```
13 {
14
15 core *a = new(core);
16
17 string source_dir = string("/home/projects/pVict0004/512/DnsData/");
18 string out_dir = "/home/projects/pVict0004/512/LES/LES_8_DB";
19 gaussian g(8);
20 a->createLES_DB(source_dir, g,out_dir );
21
22
23 delete a;
24 }
```

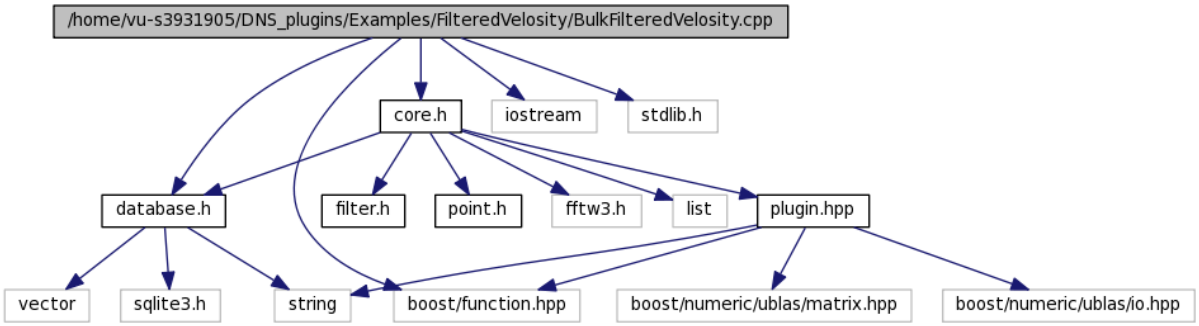
Here is the call graph for this function:


core::alocate_3Darr
Database::begin
Database::close
Database::commit
Database::end
main
core::createLES_DB
filter_base::get_fw
core::filter
filter_base::weight
core::read_file
core::indx
core::periodic_indx
Database::query
core::getdir
compare

9.1.4.2

DNS_plugins/Examples/FilteredVelosity/BulkFilteredVelosity.cpp File Reference

#include "core.h"
#include "database.h"
#include <boost/function.hpp>
#include <iostream>
#include <stdlib.h>
Include dependency graph for BulkFilteredVelosity.cpp:



9.1.5 Functions

- int main (int argc, char **argv)

9.1.6 Function Documentation

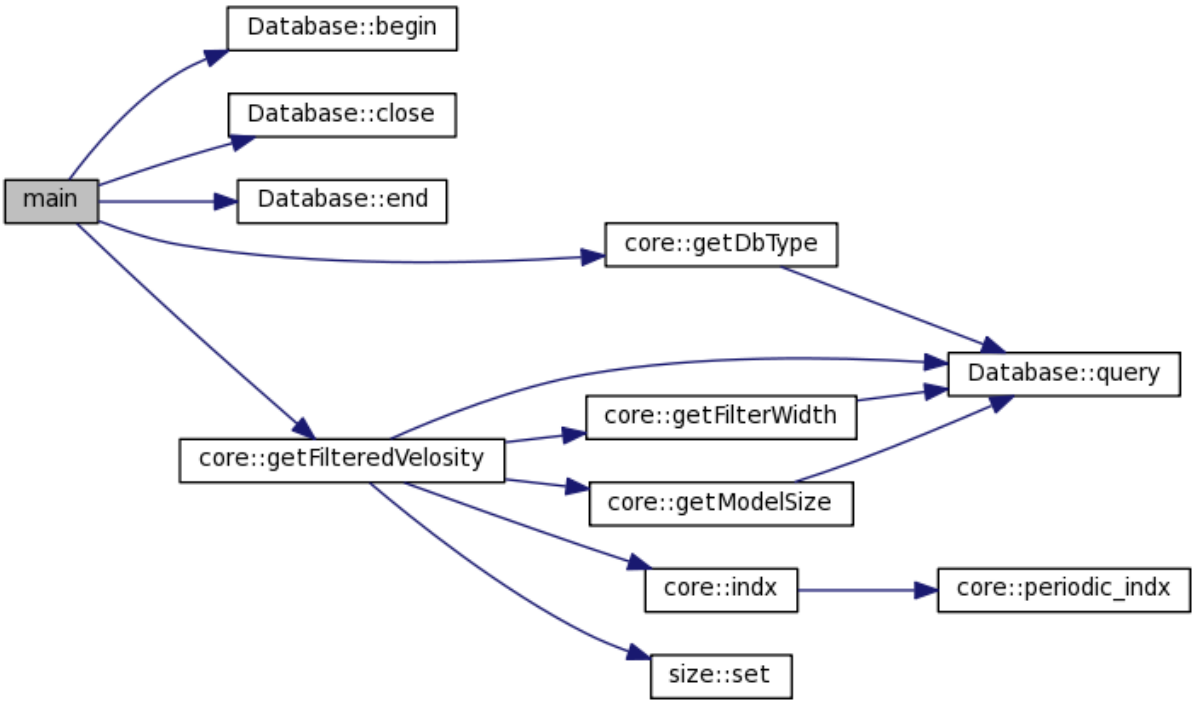
9.1.6.1 int main (int *argc*, char ** *argv*)

References Database::begin(), Database::close(), Database::end(), core::getDbType(), and core::getFilteredVelosity().

```
12 {
13
14 core *a = new(core);
15
16 Database *db =new Database((char*)"/home/projects/pVict0004/512/LES/LES_4_DB");
17
18 int y = 10;
19 int z = 10;
20 LIST_DATA ld;
21 LIST_POINTS lp;
22 LIST_DATA::iterator it;
23 for(int x=0; x< 10; x++)
24 {
25        point p(x,y,z);
26        lp.push_back(p);
27 }
```

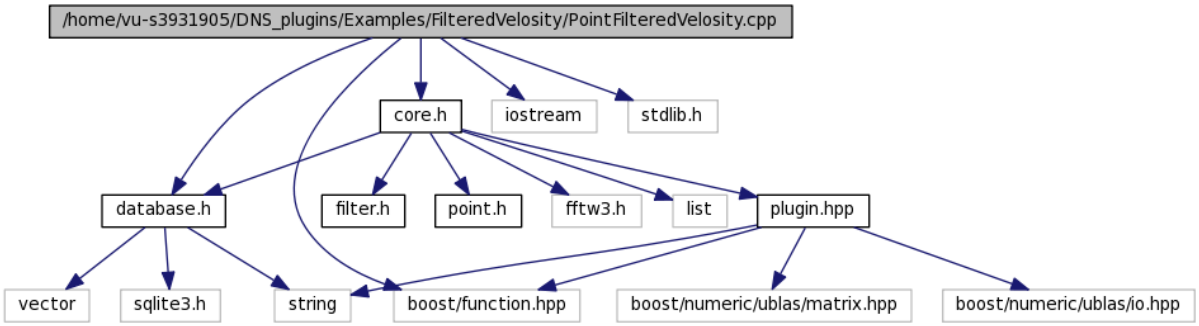
```
28
29
30 cout << a->getDbType(db) << endl;
31
32 db->begin();
33 a->getFilteredVelosity(lp,ld,db);
34 db->end();
35
36 for(it=ld.begin(); it != ld.end() ; ++it)
37 {
38 cout << " filtered u=" << (*it).p[0] ;
39 cout << " filtered v=" << (*it).p[1] ;
40 cout << " filtered w=" << (*it).p[2] ;
41 cout << " filtered u*v=" << (*it).p[0]*(*it).p[1] << endl;
42 }
43
44
45 db->close();
46 delete db;
47 delete a;
48 }
```

Here is the call graph for this function:



DNS_plugins/Examples/FilteredVelosity/PointFilteredVelosity.cpp File Reference

#include "core.h"
#include "database.h"
#include <boost/function.hpp>
#include <iostream>
#include <stdlib.h>
Include dependency graph for PointFilteredVelosity.cpp:



9.1.7 Functions

- int main (int argc, char **argv)

9.1.8 Function Documentation

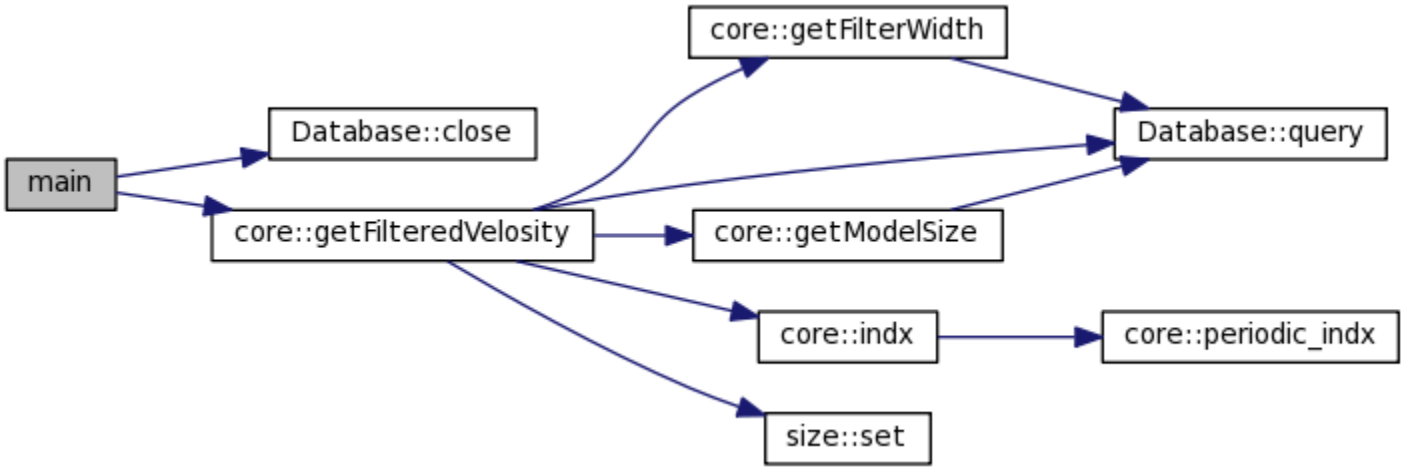
9.1.8.1 int main (int *argc*, char ** *argv*)

References Database::close(), core::getFilteredVelosity(), and FilteredData::p.

```
12 {
13
14 core *a = new(core);
15
16 Database *db =new Database((char*)"/home/projects/pVict0004/512/LES/LES_2_DB");
17
18 point p(12,24,32);
19
20 FilteredData  res = a->getFilteredVelosity(p,db);
21
22 cout << "filtered u=" << res.p[0] << endl;
23 cout << "filtered v=" << res.p[1] << endl;
24 cout << "filtered w=" << res.p[2] << endl;
25 cout << "filtered u*v=" << res.p[0]*res.p[1] << endl;
26
27 db->close();
28 delete db;
```

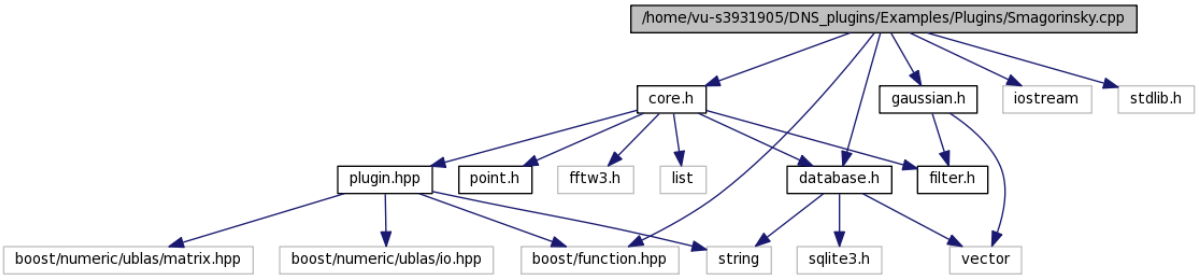
```
29 delete a;
30 }
```

Here is the call graph for this function:



DNS_plugins/Examples/Plugins/Smagorinsky.cpp File Reference

#include "core.h"
#include "database.h"
#include <boost/function.hpp>
#include "gaussian.h"
#include <iostream>
#include <stdlib.h>
Include dependency graph for Smagorinsky.cpp:



9.1.9 Functions

- int main (int argc, char **argv)

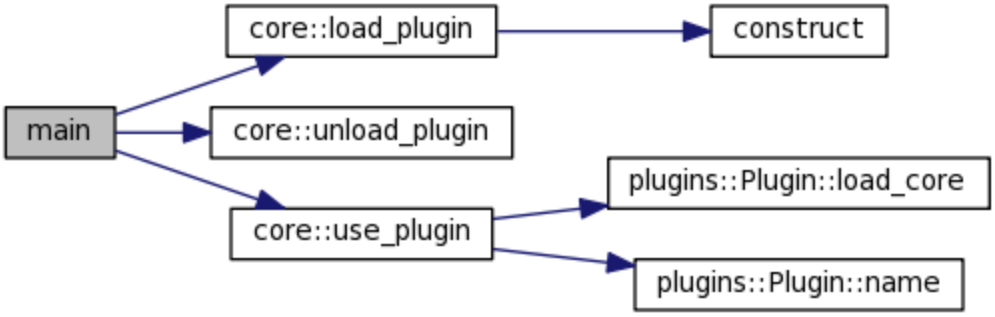
9.1.10 Function Documentation

9.1.10.1 int main (int *argc*, char ** *argv*)

References core::load_plugin(), core::unload_plugin(), and core::use_plugin().

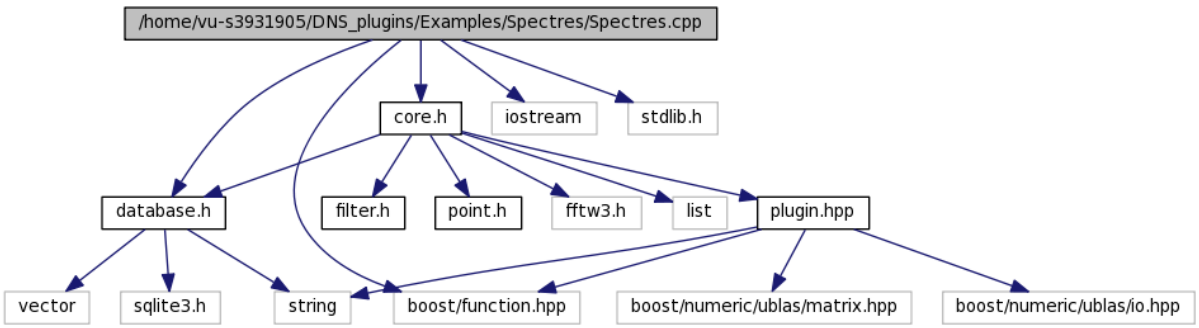
```
13 {
14
15 core *a = new(core);
16 a->load_plugin();
17 a->use_plugin();
18 a->unload_plugin();
19
20
21 delete a;
22 }
```

Here is the call graph for this function:


core::load_plugin
construct
main
core::unload_plugin
plugins::Plugin::load_core
core::use_plugin
plugins::Plugin::name

DNS_plugins/Examples/Spectres/Spectres.cpp File Reference

#include "core.h"
#include "database.h"
#include <boost/function.hpp>
#include <iostream>
#include <stdlib.h>
Include dependency graph for Spectres.cpp:



9.1.11 Functions

- int main (int argc, char **argv)

Function Documentation

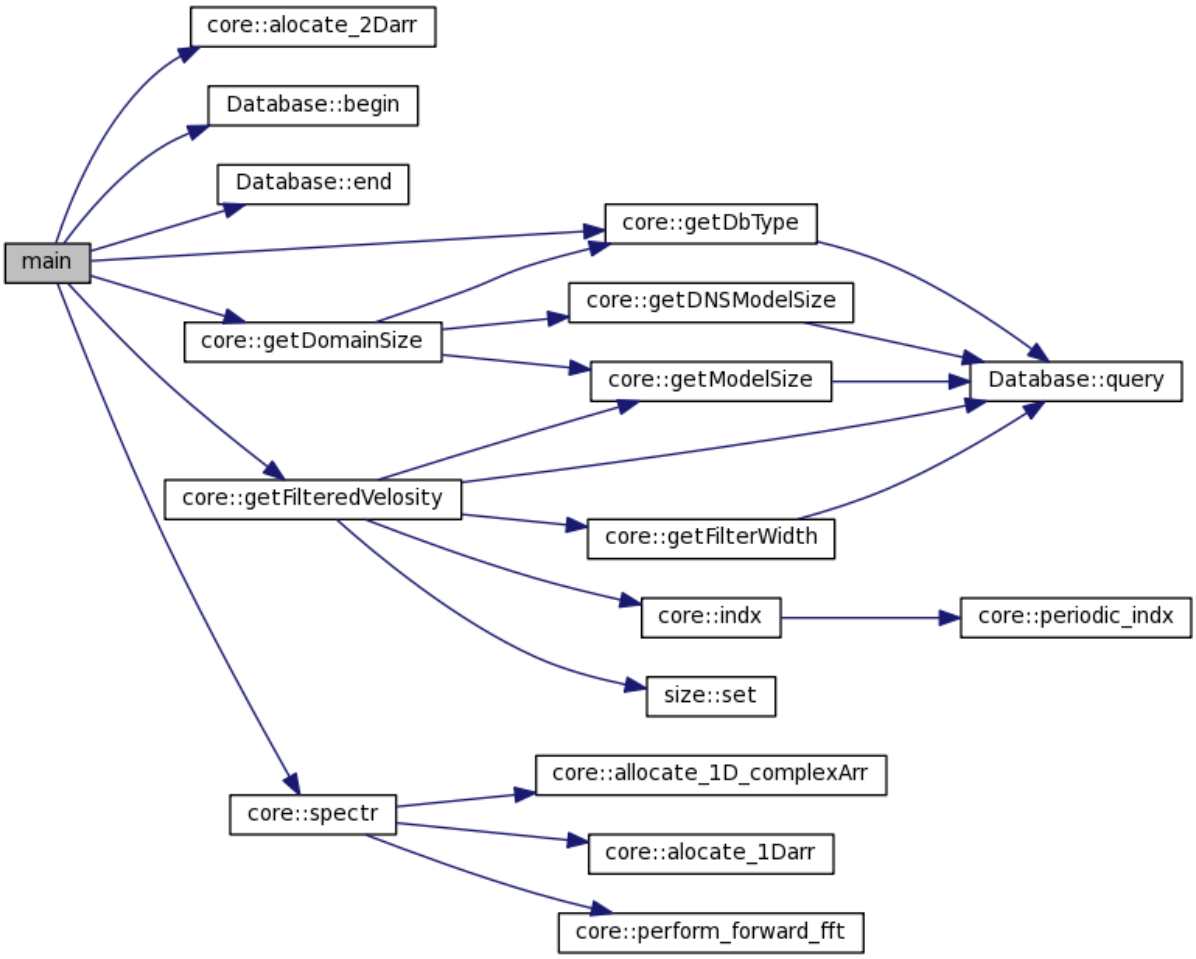
9.1.11.1 int main (int *argc*, char ** *argv*)

References core::alocate_2Darr(), Database::begin(), Database::end(), core::getDbType(), core::getDomainSize(), core::getFilteredVelosity(), size::lenX, and core::spectr().

```
12 {
13
14 core *a = new(core);
15
16 Database *db =new Database((char*)"/home/projects/pVict0004/512/LES/LES_4_DB");
17 //Database *db =new
Database((char*)"/home/projects/pVict0004/512/DnsDataDB/DNS_DB");
18
19 if (a->getDbType(db) != "LES" )
20 {
21 cout << "This is not LES database , exiting " << endl;
22 return -1;
23 }
24
25 FILE *fp;
26 string fname = "Spectr_z";
27 fp = fopen(fname.c_str(),"w");
```

```
28
29 size sz = a->getDomainSize(db);
30 cout << "sz=" << sz.lenX << " ," << sz.lenY << "," << sz.lenZ << endl;
31
32 int z = sz.lenZ/2;
33
34
35 LIST_POINTS lp;
36 LIST_DATA ld;
37 LIST_DATA::iterator it;
38 for (int x =0 ; x < sz.lenX ; x++)
39 {
40   for ( int y =0 ; y < sz.lenY ; y++)
41   {
42    point p(x,y,z);
43    lp.push_back(p);
44   }
45 }
46
47   db->begin();
48     a->getFilteredVelosity(lp,ld,db);
49   db->end();
50
51   double *in =  a->alocate_2Darr(sz.lenX,sz.lenY);
52
53   it=ld.begin();
54   int ind =0;
55   for (int x =0 ; x < sz.lenX ; x++)
56   {
57     for ( int y =0 ; y < sz.lenY ; y++)
58     {
59       in[ind++] = (*it).p[0];
60       it++;
61     }
62   }
63
64    a->spectr(sz,z,in,fp);
65
66    delete [] in;
67    fclose (fp);
68
69 }
```

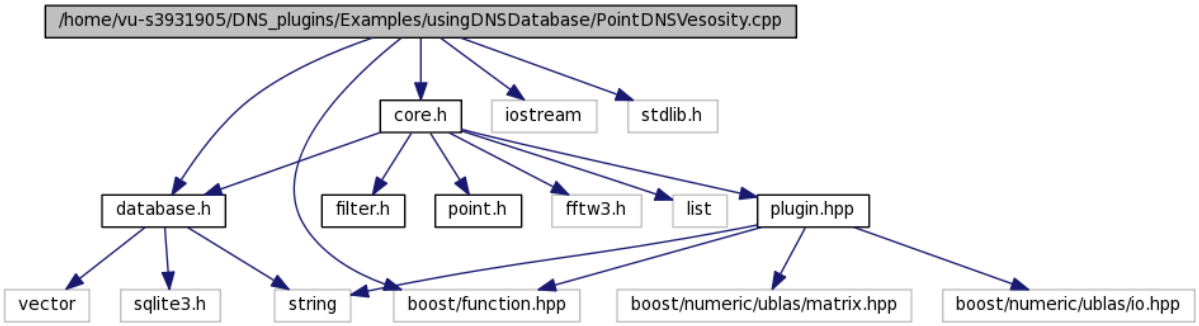
Here is the call graph for this function:


core::alocate_2Darr
Database::begin
Database::end
main
core::getDbType
core::getDNSModelSize
core::getDomainSize
core::getModelSize
Database::query
core::getFilteredVelosity
core::getFilterWidth
core::indx
core::periodic_indx
size::set
core::allocate_1D_complexArr
core::spectr
core::alocate_1Darr
core::perform_forward_fft

9.1.11.2

DNS_plugins/Examples/usingDNSDatabase/PointDNSVesosity.cpp File Reference

#include "core.h"
#include "database.h"
#include <boost/function.hpp>
#include <iostream>
#include <stdlib.h>
Include dependency graph for PointDNSVesosity.cpp:



9.1.12 Functions

- int main (int argc, char **argv)

Function Documentation

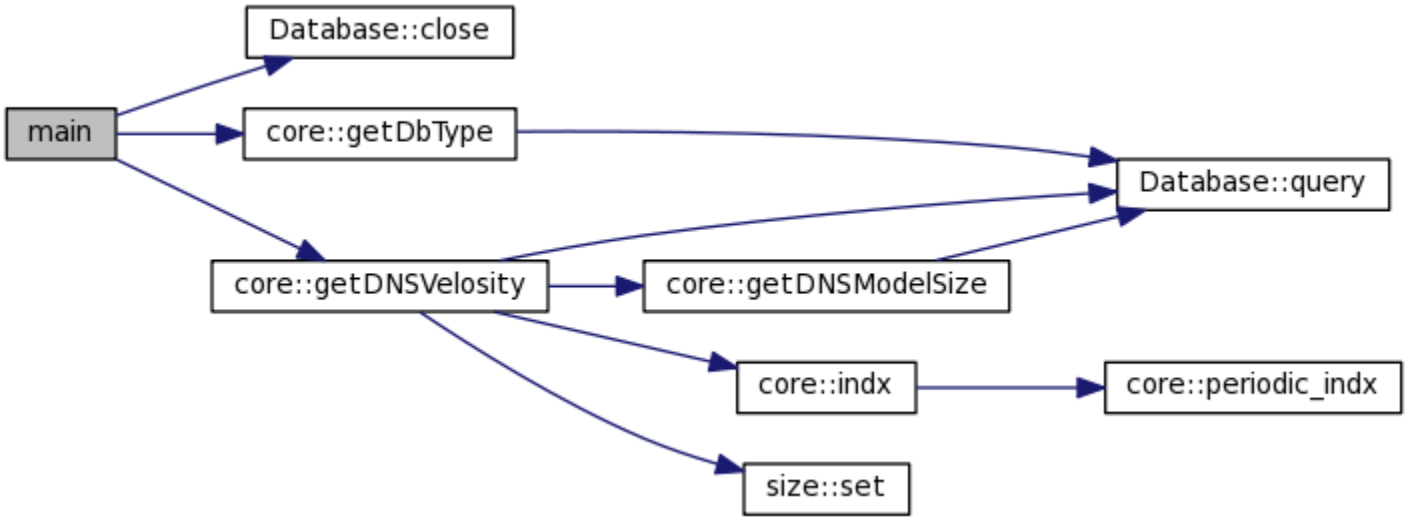
9.1.12.1 int main (int *argc*, char ** *argv*)

References Database::close(), core::getDbType(), and core::getDNSVelosity().

```
12 {
13
14 core *a = new(core);
15
16 Database *db =new
Database((char*)"/home/projects/pVict0004/512/DnsDataDB/DNS_DB");
17
18 point p(1,0,0);
19
20 double u,v,w;
21
22  cout << a->getDbType(db) << endl;
23
24  a->getDNSVelosity(p,u, v, w,db);
25
26 cout << "u=" << u << endl;
27 cout << "v=" << v << endl;
28 cout << "w=" << w << endl;
```

```
29
30 db->close();
31 delete db;
32 delete a;
33 }
```

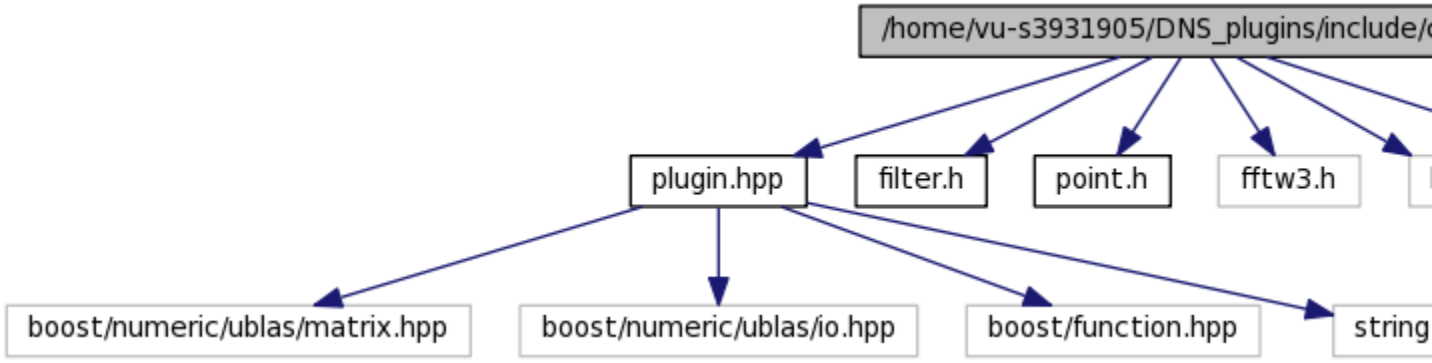
Here is the call graph for this function:



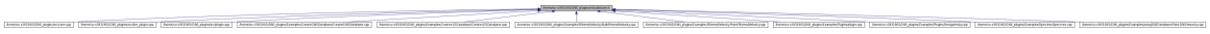
9.1.12.2

DNS_plugins/include/core.h File Reference

#include "plugin.hpp"
#include "filter.h"
#include "point.h"
#include <fftw3.h>
#include <list>
#include "database.h"
Include dependency graph for core.h:



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



9.1.13 Classes

- class core

9.1.14 Typedefs

- typedef list< point > LIST_POINTS
- typedef list< FilteredData > LIST_DATA

9.1.15 Typedef Documentation

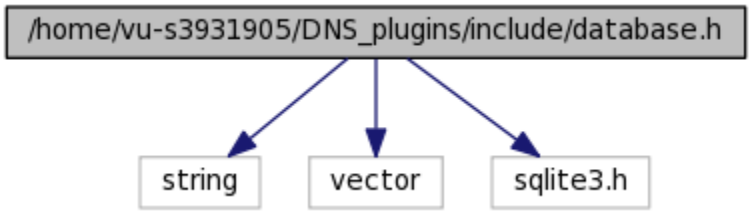
9.1.15.1 typedef list<FilteredData> LIST_DATA

9.1.15.2 typedef list<point> LIST_POINTS

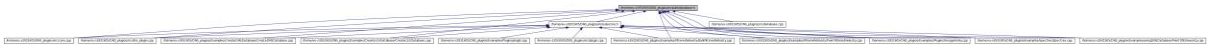
9.1.15.3

DNS_plugins/include/database.h File Reference

#include <string>
#include <vector>
#include <sqlite3.h>
Include dependency graph for database.h:



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



9.1.16 Classes

- class Database

9.1.17 Typedefs

- typedef std::vector< std::string > row
- typedef std::vector< row > table

9.1.18 Typedef Documentation

9.1.18.1 typedef std::vector<std::string> row

9.1.18.2 typedef std::vector<row> table

9.1.18.3

DNS_plugins/include/filter.h File Reference

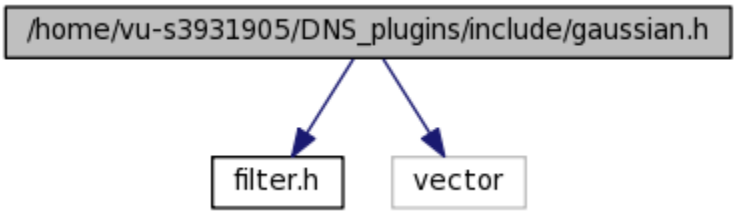
This graph shows which files directly or indirectly include this file:

9.1.19 Classes

- class filter_base

DNS_plugins/include/gaussian.h File Reference

#include "filter.h"
#include <vector>
Include dependency graph for gaussian.h:



9.1.20 Classes

- class gaussian

9.1.21 Typedefs

- typedef std::vector< std::vector< std::vector< double > > > array3D

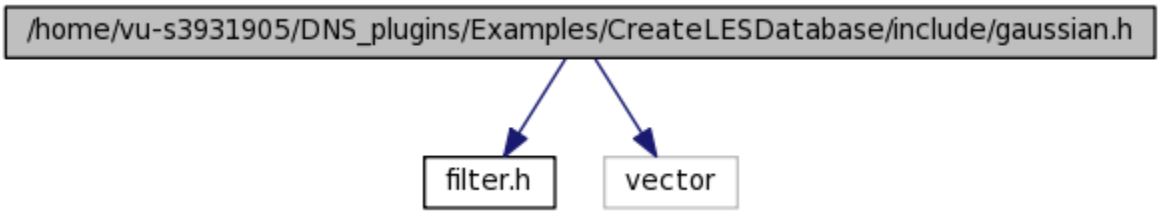
9.1.22 Typedef Documentation

9.1.22.1 typedef std::vector<std::vector<std::vector<double> > > array3D

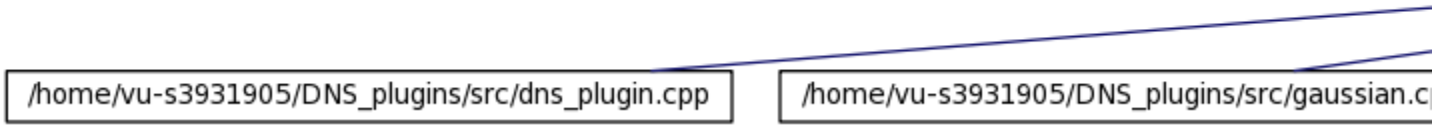
9.1.22.2

DNS_plugins/Examples/CreateLESDatabase/include/gaussian.h File Reference

#include "filter.h"
#include <vector>
Include dependency graph for gaussian.h:



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



9.1.23 Classes

- class gaussian

9.1.24 Typedefs

- typedef std::vector< std::vector< std::vector< double > > > array3D

9.1.25 Typedef Documentation

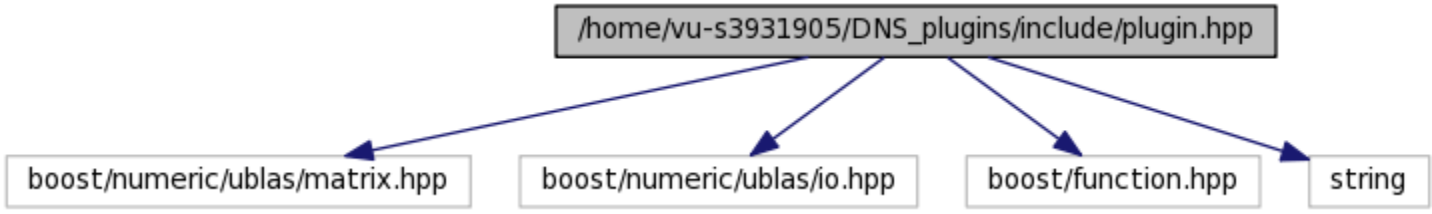
9.1.25.1 typedef std::vector<std::vector<std::vector<double> > > array3D

9.1.25.2

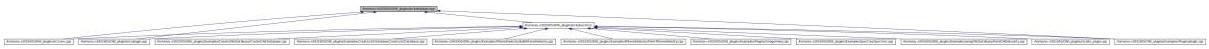
DNS_plugins/include/mainpage.dox File Reference

DNS_plugins/include/plugin.hpp File Reference

#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/io.hpp>
#include <boost/function.hpp>
#include <string>
Include dependency graph for plugin.hpp:



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



9.1.26 Classes

- class size
- class plugins::Plugin

9.1.27 Namespaces

- namespace plugins

9.1.28 Typedefs

- typedef matrix< double > Matrix

9.1.29 Enumerations

- enum order_t { ONE = 1, FOUR = 4 }

9.1.30 Typedef Documentation

9.1.30.1 typedef matrix<double> Matrix

9.1.31 Enumeration Type Documentation

9.1.31.1 enum order_t

Enumerator:

ONE

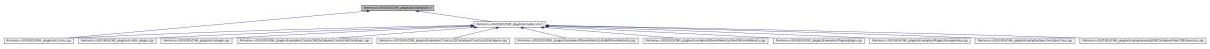
FOUR

```
16 { ONE=1,FOUR=4 } ;
```

9.1.31.2

DNS_plugins/include/point.h File Reference

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



9.1.32 Classes

- class FilteredData
- class point

9.1.33 Typedefs

- typedef double POINT3D [3]

9.1.34 Typedef Documentation

9.1.34.1 typedef double POINT3D[3]

9.1.34.2

DNS_plugins/include/task.h File Reference

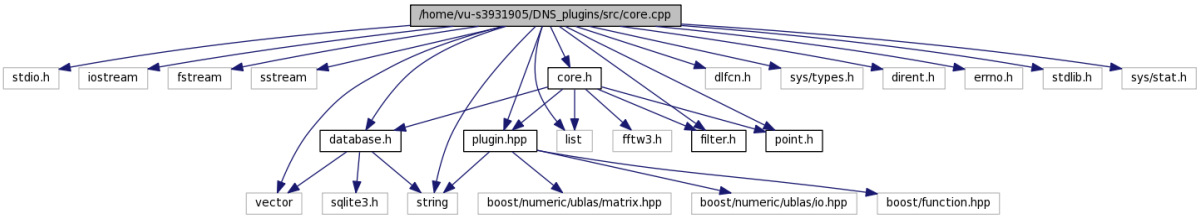
9.1.35 Classes

- struct task

DNS_plugins/src/core.cpp File Reference

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include <list>
#include <dlfcn.h>
#include <sys/types.h>
#include <dirent.h>
#include <errno.h>
#include <stdlib.h>
#include "database.h"
#include "plugin.hpp"
#include "core.h"
#include "filter.h"
#include "point.h"
#include <sys/stat.h>
```

Include dependency graph for core.cpp:



9.1.36 Typedefs

- typedef boost::function< plugins::Plugin *()> pluginConstructor

9.1.37 Functions

- bool compare (const std::string &first, const std::string &second)
- fftw_plan perform_forward_fft (int Nz, int Nx, int Ny, double *in, fftw_complex *out)

9.1.38 Typedef Documentation

9.1.38.1 typedef boost::function<plugins::Plugin* ()> pluginConstructor

---

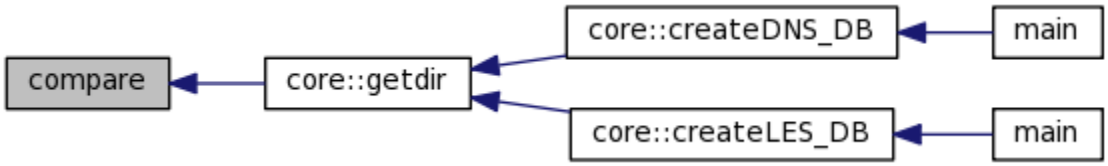
9.1.39 Function Documentation

9.1.39.1 bool compare (const std::string & *first*, const std::string & *second*)

Referenced by core::getdir().

```
26 {
27
28        int pos1=first.find('_')+1;
29        int pos2=second.find('_')+1;
30        int one = atoi(first.substr(pos1,first.length()-8).c_str());
31        int two = atoi(second.substr(pos2,second.length()-8).c_str());
32
33        if(one > two) return false;
34        return true;
35
36 }
```

Here is the caller graph for this function:

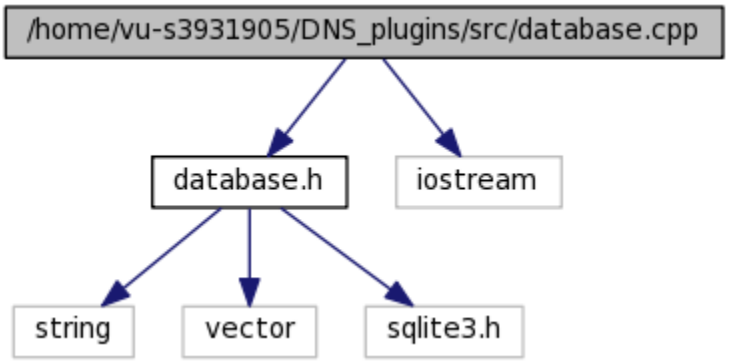


9.1.39.2 fftw_plan perform_forward_fft (int *Nz*, int *Nx*, int *Ny*, double * *in*, fftw_complex * *out*)

```
800 {
801 /*
802  create a plan and perform  discrete Fourier transform (DFT) in 3D space
803  input -pointer to real array,
804        array should be big enough to hold  Nz*Nx*Ny  double values
805  output -pointer to a complex array
806        array will hold Nz*Nx*(Ny/2 +1)  complex values
807
808  see "http://www.fftw.org/doc/Real_002ddata-DFT-Array-Format.html#Real_002ddata-
DFT-Array-Format"
809  return fftw_plan
810 */
811   fftw_plan plan_forward = fftw_plan_dft_r2c_3d ( Nz, Nx, Ny,  in, out,
FFTW_ESTIMATE );
812   fftw_execute ( plan_forward );
813   fftw_destroy_plan(plan_forward);
814   return plan_forward;
815 }
```

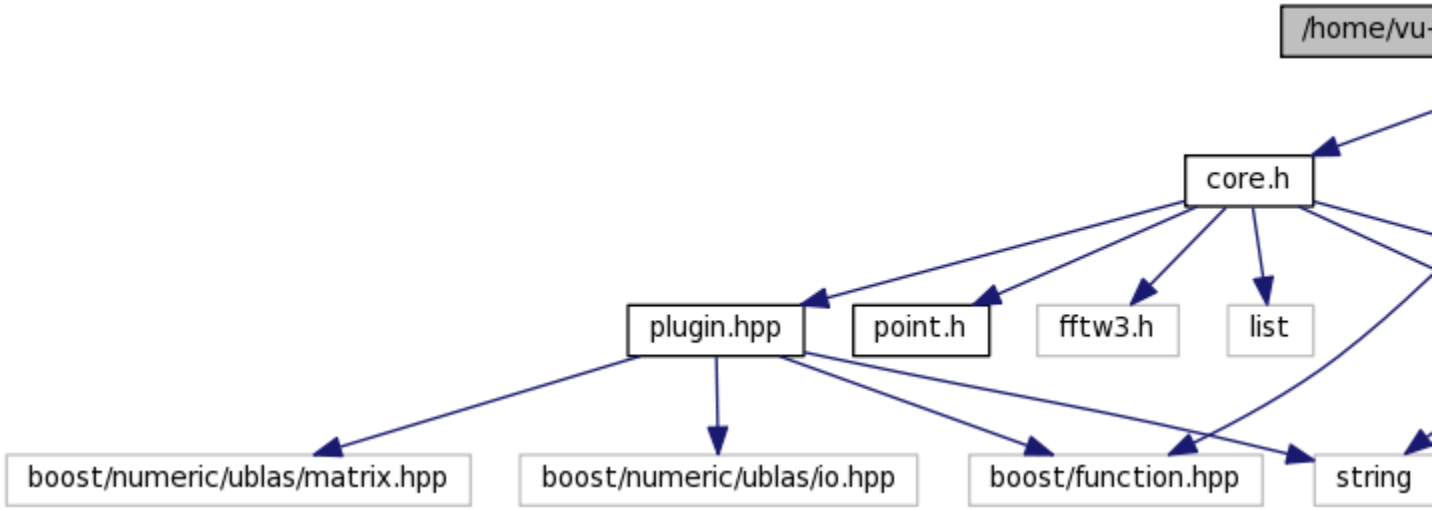
DNS_plugins/src/database.cpp File Reference

#include "database.h"
#include <iostream>
Include dependency graph for database.cpp:



DNS_plugins/src/dns_plugin.cpp File Reference

#include "core.h"
#include "database.h"
#include <boost/function.hpp>
#include "gaussian.h"
#include <iostream>
#include <stdlib.h>
Include dependency graph for dns_plugin.cpp:



9.1.40 Functions

- int main (int argc, char **argv)

9.1.41 Function Documentation

9.1.41.1 int main (int *argc*, char ** *argv*)

References Database::close(), core::getFilteredVelosity(), core::getFilterWidth(), core::getModelSize(), and FilteredData::p.

```
16 {
17 /*
18 ArgvParser cmd;
19 // init
20   cmd.setIntroductoryDescription("This is foo written by bar.");
21 //define error codes
22   cmd.addErrorCode(0, "Success");
23   cmd.addErrorCode(1, "Error");
24   cmd.setHelpOption("h", "help", "Print this help page");
25
```

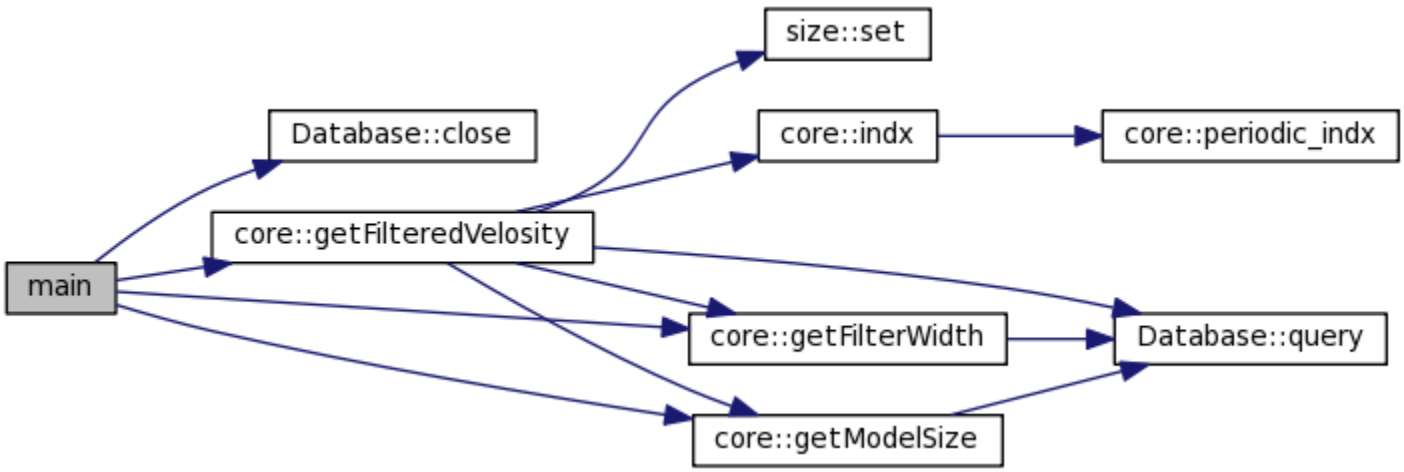
```
26  //  cmd.defineOption("version", ArgvParser::NoOptionAttribute, "Be verbose");
27   cmd.defineOption("version", "Be verbose", ArgvParser::NoOptionAttribute );
28
29
30   cmd.defineOptionAlternative("verbose","v");
31
32  // cmd.defineOption("foo", ArgvParser::OptionRequiresValue, "Fooishness. Default
value: 0");
33   cmd.defineOption("foo",  "Fooishness. Default value:
0",ArgvParser::OptionRequiresValue );
34
35   cmd.defineOption("createDb" );
36   cmd.defineOption("sp", "", ArgvParser::OptionRequired);
37
38
39  // finally parse and handle return codes (display help etc...)
40    int result = cmd.parse(argc, argv);
41
42    if (result != ArgvParser::NoParserError)
43    {
44        cout << cmd.parseErrorDescription(result);
45        exit(1);
46    }
47
48    // now query the parsing results
49    if (cmd.foundOption("foo"))
50    {
51     // string = cmd.optionValue("foo");
52        cout << cmd.optionValue("foo") << endl;
53    }
54    if(cmd.foundOption("createDb"))
55    {
56        cout << cmd.optionValue("createDB") << endl;
57    }
58
59
60 return 0;
61 */
62
63 core *a = new(core);
64
65 string source_dir = string("/home/projects/pVict0004/512/DnsData/");
66 string out_dir = "/home/projects/pVict0004/512/LES/";
67
68 /*
69 out_dir += "LES_2_DB";
```

```
70 gaussian g(2);
71 a->createLES_DB(source_dir, g,out_dir );
72 */
73
74 /*
75 out_dir += "LES_4_DB";
76 gaussian g(4);
77 a->createLES_DB(source_dir, g,out_dir );
78 */
79
80 /*
81 out_dir += "LES_6_DB";
82 gaussian g(6);
83 a->createLES_DB(source_dir, g,out_dir );
84 */
85 /*
86 out_dir += "LES_8_DB";
87 gaussian g(8);
88 a->createLES_DB(source_dir, g,out_dir );
89 */
90
91 /*
92 out_dir += "LES_16_DB";
93 gaussian g(16);
94 a->createLES_DB(source_dir, g,out_dir );
95 return 0;
96 */
97
98 Database *db;
99 db = new Database((char*)"/home/projects/pVict0004/512/LES/LES_2_DB");
100
101 cout << "fw=" << a->getFilterWidth(db) <<endl;
102 cout << "sz=" << a->getModelSize(db) <<endl;
103
104 point p(12,24,32);
105
106 FilteredData  res = a->getFilteredVelosity(p,db);
107
108 cout << "u=" << res.p[0] << endl;
109
110 int y = 10;
111 int z = 10;
112 LIST_DATA ld;
113 LIST_POINTS lp;
114 LIST_DATA::iterator it;
115 for(int x=0; x< 10; x++)
```

```
116 {
117 point p(x,y,z);
118 lp.push_back(p);
119 }
120 a->getFilteredVelosity(lp,ld,db);
121
122 for(it=ld.begin(); it != ld.end() ; ++it)
123 {
124  cout << "u=" << (*it).p[0] << endl;
125 }
126
127
128
129
130
131 /*
132 table res = db->query((char*)"SELECT * FROM data WHERE ind=2;");
133
134 table::iterator it;
135 for(it = res.begin(); it < res.end(); ++it)
136 {
137         row  rw = *it;
138         cout << "Values: (ind=" << rw.at(0) <<
139                         ", u=" << rw.at(1) <<
140                         ", v=" << rw.at(2) <<
141                         ", w=" << rw.at(3) <<
142                         ")" << endl;
143         //cout << "Values: (A=" << rw.at(0) << ", B=" << rw.at(1) << ")" << endl;
144 }
145 */
146 db->close();
147
148 //a->load_plugin();
149 //a->use_plugin();
150 //a->unload_plugin();
151
152 //a->model_stress();
153
154
155 /*
156 Database *db;
157 db = new Database((char*)"/home/vu-s3931905/DNS_plugins/src/Database.sqlite");
158
159 db->query((char*)"CREATE TABLE a (a INTEGER, b INTEGER);");
160 db->query((char*)"INSERT INTO a VALUES(1, 2);");
161 db->query((char*)"INSERT INTO a VALUES(5, 4);");
```

```
162 table::iterator it;
163
164
165
166 table res = db->query((char*)"SELECT a, b FROM a;");
167
168 for(it = res.begin(); it < res.end(); ++it)
169 {
170         row  rw = *it;
171         cout << "Values: (A=" << rw.at(0) << ", B=" << rw.at(1) << ")" << endl;
172 }
173
174 db->close();
175 */
176
177
178
179
180 }
```

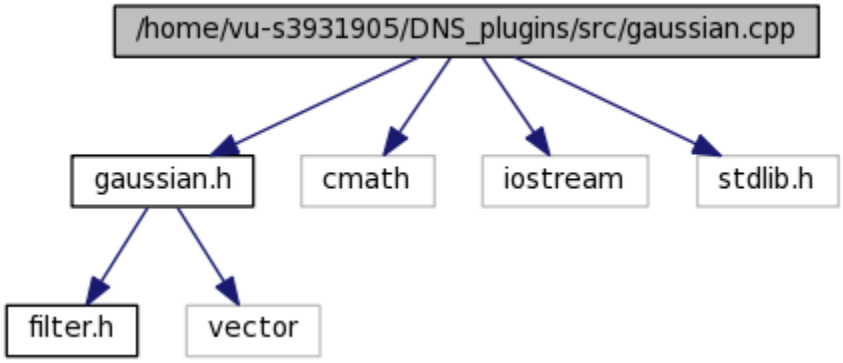
Here is the call graph for this function:



9.1.41.2

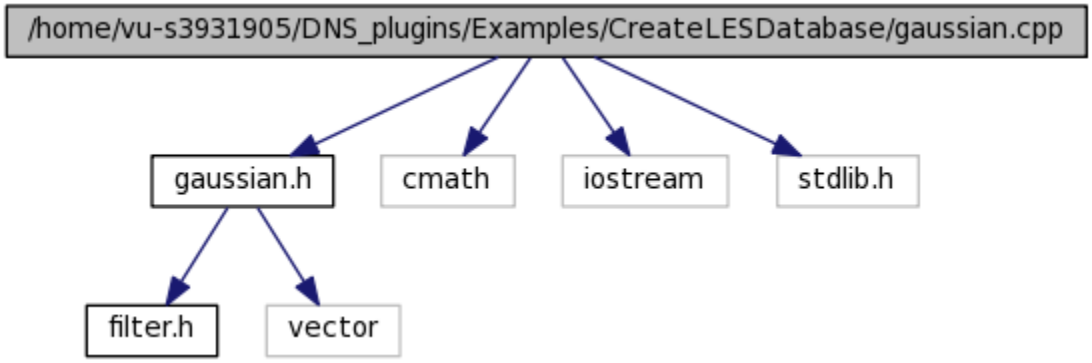
DNS_plugins/src/gaussian.cpp File Reference

#include "gaussian.h"
#include <cmath>
#include <iostream>
#include <stdlib.h>
Include dependency graph for gaussian.cpp:



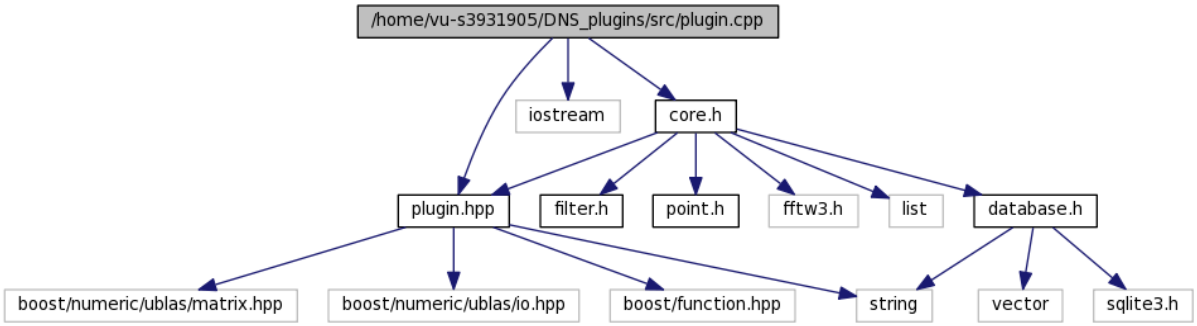
DNS_plugins/Examples/CreateLESDatabase/gaussian.cpp File Reference

#include "gaussian.h"
#include <cmath>
#include <iostream>
#include <stdlib.h>
Include dependency graph for gaussian.cpp:



DNS_plugins/src/plugin.cpp File Reference

#include "plugin.hpp"
#include <iostream>
#include "core.h"
Include dependency graph for plugin.cpp:



9.1.42 Classes

- class plugins::Smagorinsky

9.1.43 Namespaces

- namespace plugins

9.1.44 Functions

- plugins::Plugin * construct ()

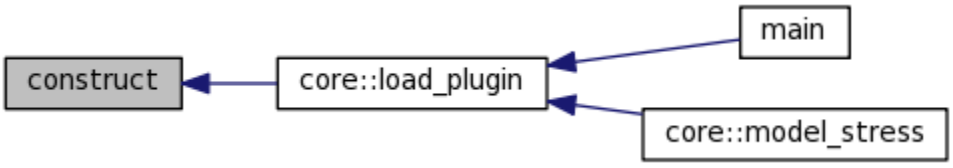
9.1.45 Function Documentation

9.1.45.1 plugins::Plugin* construct ()

Referenced by core::load_plugin().

```
83     {
84         return new plugins::Smagorinsky();
85     }
```

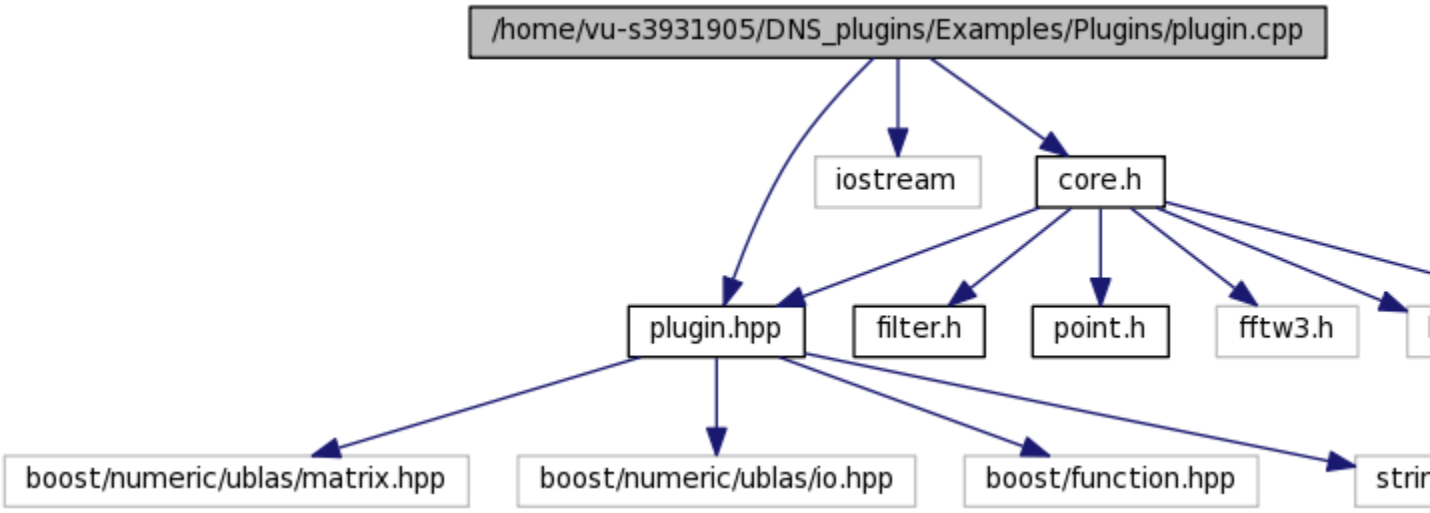
Here is the caller graph for this function:



9.1.45.2

DNS_plugins/Examples/Plugins/plugin.cpp File Reference

#include "plugin.hpp"
#include <iostream>
#include "core.h"
Include dependency graph for plugin.cpp:



9.1.46 Classes

- class plugins::Smagorinsky

9.1.47 Namespaces

- namespace plugins

9.1.48 Functions

- plugins::Plugin * construct ()

9.1.49 Function Documentation

9.1.49.1 plugins::Plugin* construct ()

```
83    {
84        return new plugins::Smagorinsky();
85    }
```

10 Index