NP Transition Matrix

Generated by Doxygen 1.9.8

Chapter 1

Folder instructions

This directory collects the source code of the original programs and the development folders.

1.1 Instructions

This folder is intended to store the sources of the original code and the C++ implementation of the project. Distributing and changing the source is possible under the terms of the GNU GPLv3 license (see *License* below). Use of this code and of any derived implementation should credit the original authors. Scientific publications should do so by citing the following references:

- Saija et al. 2001, ApJ, 559, 993, DOI:10.1086/322350
- Borghese, Denti, Saija 2007, Scattering from Model Nonspherical Particles (ISBN 978-3-540-37413-8), DOI↩ :10.1007/978-3-540-37414-5

Instructions on how to set up and use the code are instead given in the project's `build` folder.

1.2 License

Chapter 2

Folder instructions

This directory contains scripts and tools to evaluate the code functionality.

2.1 Instructions

The code migration stage can be considered successfully fulfilled with the solution of the single sphere and of the cluster of spheres cases in *C++*. To test the reliability of the code, the *C++* version needs to produce consistent output with respect to the original *FORTRAN* code. Since the output files are generally too large for manual inspection and they can be affected by different approximations or numeric noise, a series of scripts designed to compare the two versions and to pin-point possible differences is provided in this folder. The scripts are written in *python*, therefore they need an available *python* environment to work.

2.2 Create models from YAML input

The execution of `NP_TMcode` applications requires machine readable input files to describe the particle model and the radiation field properties. This files are ASCII files that can be edited with common text editors, but they obey a strict format which complies with the requirements of the original code implementation. In order to make the model creation easier, `NP_TMcode` provides a model editor script named `model_maker.py`. This script reads a model description from a `YAML` file and creates the necessary input configuration for the `NP_TMcode` applications. To run the script, just issue:

```
$PATH_TO_SCRIPT/model_maker.py YAML_CONFIGURATION_FILE
```

where `YAML_CONFIGURATION_FILE` must be a valid `YAML` model description (examples are given in the `test_data` folder). More details are given issuing:

```
$PATH_TO_SCRIPT/model_maker.py –help
```

2.3 Compare the results of FORTRAN and C++ codes

1. Follow the instructions to build and run the *FORTRAN* and *C++* versions of the code.
2. Run the `pycompare.py` script providing the following minimal arguments:

   $PATH_TO_SCRIPT/pycompare.py –ffile FORTRAN_OUTPUT –cfile C++_OUTPUT

(The above assumes that `PATH_TO_SCRIPT` is a variable that expands to the path were the script is located). The required output files are called by default `OSPH` and `OCLU` by *FORTRAN* and `c_OSPH` and `c_OCLU` by *C++*, depending on whether the sphere or the cluster case was executed.

1. Check the output of the script to verify that it detects 0 errors and finishes in a `SUCCESS` state.
2. In case of necessity, add the `--html` argument to produce an *HTML* log showing the possible differences and a classification of their severity.
3. Issuing:

   $PATH_TO_SCRIPT/pycompare.py –help

or just:

   $PATH_TO_SCRIPT/pycompare.py

will print a help screen, giving a brief explanation of all the possible options.

2.4 Estimate the recommended orders for running a model

This script applies the Wiscombe criterion and its modifications to estimate the recommended internal and external orders to run a model. To use it, issue:

   $PATH_TO_SCRIPT/pywiscombe.py –li|le –wave=WAVELENGTH –rad=RADIUS

Wavelength and radius must be expressed in meters, with the radius being equal to the minimum radius encircling the particle. More details are given with:

   $PATH_TO_SCRIPT/pywiscombe.py –help

2.5 Estimate the execution time from a terminal log

Performance estimates can be obtained from the code logging system, assuming the uses chose to save the terminal output to some log file. To obtain the time spent by the code in performing a specific operation, the syntax is:

$PATH_TO_SCRIPT/pytiming.py –logname=LOG_FILE [–filter=FILTER –threads=NUM_THREADS]

where LOG_FILE must be the name of a file containing the output that would normally go to terminal, FILTER must be the starting character sequence of the log line identifying the operation that should be taken into account, and NUM_THREADS is the number of processes that were used to perform the whole calculation loop. In case no filter is given, the script provides an estimate of the total amount of time spent in doing the calculation. This estimate, however, is known not to be reliable, because it ignores thread concurrency effects. A more accurate estimate of the total time spent in executing the code is always saved in a file named c_timing.log.

2.6 Extract parameters from the code output

The legacy and HDF5 outputs of NP_TMcode applications contain all the values computed by the code. However, in many cases the user is interested in extracting plots of cross-sections, forces and torques as functions of wavelength. The parse_output.py script performs this task (currently only on the legacy output). To use the script after running a model, issue:

$PATH_TO_SCRIPT/parse_output.py –in CODE_RESULT_FILE –out OUTPUT_BASE_NAME –app=[SPH|CLU|INCLU]

This will parse the output of np_sphere (app=SPH), np_cluster (app=CLU) or np_inclusion (app=INCLU) and put the results in CSV text files, named after the given OUTPUT_BASE_NAME, but with distinguishing suffixes such as _ics for integrated cross-sections, _dcs for differential ones, _irp for integrated radiation pressure forces, etc. More details are given issuing:

$PATH_TO_SCRIPT/parse_output.py –help

2.7 Extract the dynamic range of a complex matrix

This script examines a complex matrix and extracts histograms of the orders of magnitude of the real parts, the imaginary parts and the absolute values. Optionally, if matplotlib is available, a plot of the dynamic range is drawn. To use this script, issue:

$PATH_TO_SCRIPT/pydynrange.py MATRIX_FILE

where MATRIX_FILE must be a CSV representation of the matrix. More details are given issuing:

$PATH_TO_SCRIPT/pydynrange.py –help

2.8 License

Chapter 3

Namespace Index

3.1 Namespace List

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

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

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

Chapter 7

Namespace Documentation

7.1 inertia Namespace Reference

Compute the momentum of inertia for a model particle.

Functions

- main ()
  Main execution code.
- filling_factor (stypes, i_tot_cm, config)
  Compute the particle's filling factor.
- get_centers (config)
  Create a dictionary of sphere types.
- get_cm_inertia_tensor (masses, stypes, centers)
  Compute the tensor of inertia of an aggregate with respect to the center of mass.
- get_inertia_tensor (masses, stypes, centers)
  Compute the tensor of inertia of an aggregate with respect to origin of coordinates.
- get_particle_mass (stypes, config)
  Compute the total particle mass.
- get_sphere_mass (isphere, stypes, config)
  Compute the mass of a given sphere.
- get_sphere_type_mass (itype, stypes, config)
  Compute the mass of a given type of sphere.
- get_total_volume (stypes, config)
  Compute the total volume of the spheres.
- get_types (config)
  Create a dictionary of sphere types.
- index_of (element, array)
  Get the index of an element in an array.
- ingest_line (line)
  Tranform a line in a sequence of string elements.
- parse_arguments ()
  Parse the command line arguments.
- **print_help** ()
  Print a command-line help summary.

7.1.1 Detailed Description

The complete description of a model particle requires knowledge of some properties, such as filling factor and symmetry. These can be estimated by comparing the model particle with its ellipsoiod of inertia. The purpose of this script is to perform such comparison on NP_TMcode model particles.

7.1.2 Function Documentation

7.1.2.1 filling_factor()

```
inertia.filling_factor (
              stypes,
              i_tot_cm,
              config )
```

In this application the filling factor is defined as the ratio of the sum of the volumes of the spheres that compose the particle with respect to the volume of a homogeneous ellipsoid that has the same rotational properties as the particle.

Parameters

stypes	`dict` Dictionary of sphere types.
i_tot_cm	`numpy.ndarray` Inertia tensor of the particle with respect to the center of mass.
config	`dict` Configuration dictionary.

Returns

ff: `float` The particle's filling factor.

7.1.2.2 get_centers()

```
inertia.get_centers (
              config )
```

This function opens a NP_TMcode scatterer configuration file and creates sphere type dictionary from it.

Parameters

config	`dict` Configuration dictionary.

Returns

centers: `dict` Sphere centers dictionary.

7.1.2.3 get_cm_inertia_tensor()

```
inertia.get_cm_inertia_tensor (
              masses,
```

```
            stypes,
            centers )
```

Parameters

masses	`numpy.ndarray` Array of masses in the aggregate.
stypes	`dict` Dictionary of sphere types.
centers	`dict` Dictionary containing the positions of the spheres.

Returns

Itot: `numpy.ndarray` The total tensor of inertia with respect to the center of mass.

7.1.2.4 get_inertia_tensor()

```
inertia.get_inertia_tensor (
            masses,
            stypes,
            centers )
```

Parameters

masses	`list-like` Array of masses in the aggregate.
stypes	`dict` Dictionary of sphere types.
centers	`dict` Dictionary containing the positions of the spheres.

Returns

Itot: `numpy.ndarray` The total tensor of inertia with respect to the reference frame.

7.1.2.5 get_particle_mass()

```
inertia.get_particle_mass (
            stypes,
            config )
```

Parameters

stypes	`dict` Sphere types dictionary.
config	`dict` Configuration dictionary.

Returns

mass: `float` The mass of the particle, expressed in kg.

7.1.2.6 get_sphere_mass()

```
inertia.get_sphere_mass (
```

```
                isphere,
                stypes,
                config )
```

Parameters

isphere	`int` ID of the sphere in the aggregate (base 1).
stypes	`dict` Sphere types dictionary.
config	`dict` Configuration dictionary.

Returns

mass: `float` The mass of the sphere, expressed in kg.

7.1.2.7 get_sphere_type_mass()

```
inertia.get_sphere_type_mass (
                itype,
                stypes,
                config )
```

Parameters

itype	`int` ID of the sphere type (base 1).
stypes	`dict` Sphere types dictionary.
config	`dict` Configuration dictionary.

Returns

type_mass: `float` The mass of the type of sphere, expressed in kg.

7.1.2.8 get_total_volume()

```
inertia.get_total_volume (
                stypes,
                config )
```

Parameters

stypes	`dict` Sphere types dictionary.
config	`dict` Configuration dictionary.

Returns

vtot: `float` The sum of the volumes of the spheres.

7.1.2.9 get_types()

```
inertia.get_types (
              config )
```

This function opens a NP_TMcode scatterer configuration file and creates sphere type dictionary from it.

Parameters

config	`dict` Configuration dictionary.

Returns

stypes: `dict` Sphere types dictionary.

7.1.2.10 index_of()

```
inertia.index_of (
              element,
              array )
```

Parameters

element	Anything.
array	`array-like` Array with the same type of the element sought for.

Returns

index: `int` 0-based index of the element.

7.1.2.11 ingest_line()

```
inertia.ingest_line (
              line )
```

Parameters

line	`string` Input file line

Returns

result: List of `string` elements.

7.1.2.12 main()

```
inertia.main ( )
```

`main()` is the function that handles the creation of the script configuration and the execution of the comparison. It returns an integer value corresponding to the script exit code.

Returns

errors: `int` Number of runtime errors (0 means successful run).

7.1.2.13 parse_arguments()

```
inertia.parse_arguments ( )
```

The script behaviour can be modified through a set of mandatory and optional arguments. Mandatory arguments are those required to execute a meaningful parsing and they are limited to the names of the files that need to be read and the operational mode ("edfb" for scattering model file, "geom" for geometry model file).

Returns

config: `dict` A dictionary containing the script configuration.

7.2 model_maker Namespace Reference

Script to build models from YAML configuration files.

Functions

- main ()
  Main execution code.
- interpolate_constants (sconf)
  Populate the dielectric constant data via interpolation.
- load_model (model_file, rnd_attempts)
  Create tha calculation configuration structure from YAML input.
- match_grid (sconf)
  Populate the dielectric constant data matching a grid.
- parse_arguments ()
  Parse the command line arguments.
- **print_help** ()
  Print a command-line help summary.
- print_model_summary (scatterer, geometry)
  Print a summary of model properties.
- random_aggregate (scatterer, geometry, seed, max_rad, max_attempts=100)
  Generate a random cluster aggregate from YAML configuration options.
- random_compact (scatterer, geometry, seed, max_rad)
  Generate a random compact cluster from YAML configuration options.
- test_system_resources (model, gconf, sconf)
  Perform a preliminary test of the resources required by the model.
- write_legacy_gconf (conf)
  Write the geometry configuration dictionary to legacy format.
- write_legacy_sconf (conf)
  Write the scatterer configuration dictionary to legacy format.
- write_obj (scatterer, geometry, max_rad)
  Export the model to a set of OBJ files for 3D visualization.

Variables

- bool **allow_3d** = True

  3D software generation capability flag.
- **exit_code** = main()

  Exit code (0 for success)

7.2.1 Detailed Description

Script to assist in the creation of model input files starting from a YAML descriptor.

The script requires python3.

7.2.2 Function Documentation

7.2.2.1 interpolate_constants()

```
model_maker.interpolate_constants (
              sconf )
```

Parameters

sconf	`dict` Scatterer configuration dictionary.

Returns

result: `int` An exit code (0 if successful).

7.2.2.2 load_model()

```
model_maker.load_model (
              model_file,
              rnd_attempts )
```

Parameters

model_file	`str` Full path to the YAML input file.
rnd_attempts	`int` Limiting number of attempts to randomly place a sphere.

Returns

sconf, gconf: `tuple` A dictionary tuple for scatterer and geometric configurations.

7.2.2.3 main()

```
model_maker.main ( )
```

main() is the function that handles the creation of the code configuration. It returns an integer value as exit code, using 0 to signal successful execution.

Returns

result: int Exit code (0 = SUCCESS).

7.2.2.4 match_grid()

```
model_maker.match_grid (
              sconf )
```

Important note: if the configuration requests that more than one optical constants file should be used, all the files must provide their constants for the same vector of wavelengths.

Parameters

sconf	dict Scatterer configuration dictionary.

Returns

result: int An exit code (0 if successful).

7.2.2.5 parse_arguments()

```
model_maker.parse_arguments ( )
```

The script behaviour can be modified through a set of optional arguments. The purpose of this function is to parse the command line in search for such arguments and prepare the execution accordingly.

Returns

config: dict A dictionary containing the script configuration.

7.2.2.6 print_model_summary()

```
model_maker.print_model_summary (
              scatterer,
              geometry )
```

This function provides a summary of useful information concerning the radii of the particle monomers and of the equivalent mass sphere, to assist in the selection of the proper starting orders.

Parameters

scatterer	dict A dictionary for the scatterer configuration.
geometry	dict A dictionary for the geometry configuration.

7.2.2.7 random_aggregate()

```
model_maker.random_aggregate (
              scatterer,
              geometry,
              seed,
              max_rad,
              max_attempts = 100 )
```

This function generates a random aggregate of spheres using radial ejection in random directions of new spheres until they become tangent to the outermost sphere existing in that direction. The result of the generated model is directly saved in the parameters of the scatterer and geometry configuration dictionaries.

Parameters

scatterer	`dict` Scatterer configuration dictionary (gets modified)
geometry	`dict` Geometry configuration dictionary (gets modified)
seed	`int` Seed for the random sequence generation
max_rad	`float` Maximum allowed radial extension of the aggregate
max_attempts	`int` Maximum number of attempts to place a particle in any direction

Returns

result: `int` Function exit code (0 for success, otherwise number of spheres that could not be placed)

7.2.2.8 random_compact()

```
model_maker.random_compact (
              scatterer,
              geometry,
              seed,
              max_rad )
```

This function generates a random aggregate of spheres using the maximum compactness packaging to fill a spherical volume with given maximum radius, then it proceeds by subtracting random spheres from the outer layers, until the aggregate is reduced to the desired number of spheres. The function can only be used if all sphere types have the same radius. The result of the generated model is directly saved in the parameters of the scatterer and geometry configuration dictionaries.

Parameters

scatterer	`dict` Scatterer configuration dictionary (gets modified)
geometry	`dict` Geometry configuration dictionary (gets modified)
seed	`int` Seed for the random sequence generation
max_rad	`float` Maximum allowed radial extension of the aggregate

Returns

result: `int` Function exit code (0 for success, otherwise error code)

7.2.2.9 test_system_resources()

```
model_maker.test_system_resources (
              model,
              gconf,
              sconf )
```

The resolution of models requires the availability of memory resources that increase as a function of the model complexity. This function aims at evaluating the complexity of the model and estimate whether the declared system resources are sufficient to run the calculation.

Parameters

model	`dict` Model description dictionary.
gconf	`dict` Geometry description dictionary.
sconf	`dict` Scattering description dictionary.

7.2.2.10 write_legacy_gconf()

```
model_maker.write_legacy_gconf (
              conf )
```

Parameters

conf	`dict` Geometry configuration dictionary.

Returns

result: `int` An exit code (0 if successful).

7.2.2.11 write_legacy_sconf()

```
model_maker.write_legacy_sconf (
              conf )
```

Parameters

conf	`dict` Scatterer configuration dictionary.

Returns

result: `int` An exit code (0 if successful).

7.2.2.12 write_obj()

```
model_maker.write_obj (
              scatterer,
```

```
        geometry,
        max_rad )
```

This function exports the model as a single OBJ file, containing the information to visualize the particle with 3D software tools. The model file is associated with a MTL material libray file, used to assign colors to spheres of different type.

Parameters

scatterer	`dict` Scatterer configuration dictionary (gets modified)
geometry	`dict` Geometry configuration dictionary (gets modified)
max_rad	`float` Maximum allowed radial extension of the aggregate

7.3 parse_output Namespace Reference

Script to extract physical quantities from NPTMcode output files.

Functions

- main ()
  Main execution code.
- parse_arguments ()
  Parse the command line arguments.
- parse_legacy_oclu (config)
  Parse a legacy output file of np_cluster.
- parse_legacy_oinclu (config)
  Parse a legacy output file of np_inclusion.
- parse_legacy_osph (config)
  Parse a legacy output file of np_sphere.
- **print_help** ()
  Print a command-line help summary.

7.3.1 Detailed Description

This script is intended to assist users in the inspection of results. It can be used to extract physical quantities, such as radiation pressure forces and cross-sections from the standard code output files and save the data in machine readable CSV files.

The script execution requires python3.

7.3.2 Function Documentation

7.3.2.1 main()

```
parse_output.main ( )
```

`main()` is the function that handles the creation of the script configuration and the execution of the comparison. It returns an integer value corresponding to the number of detected error-level inconsistencies.

Returns

errors: `int` Number of runtime errors (0 means successful run).

7.3.2.2 parse_arguments()

```
parse_output.parse_arguments ( )
```

The script behaviour can be modified through a set of mandatory and optional arguments. Mandatory arguments are those required to execute a meaningful parsing and they are limited to the names of the files that need to be read and written. The other arguments affect the format and the data fields that are sought for.

Returns

config: `dict` A dictionary containing the script configuration.

7.3.2.3 parse_legacy_oclu()

```
parse_output.parse_legacy_oclu (
              config )
```

Parameters

config	`dict` A dictionary containing the script configuration.

Returns

errors: `int` The number of encountered errors.

7.3.2.4 parse_legacy_oinclu()

```
parse_output.parse_legacy_oinclu (
              config )
```

Parameters

config	`dict` A dictionary containing the script configuration.

Returns

errors: `int` The number of encountered errors.

7.3.2.5 parse_legacy_osph()

```
parse_output.parse_legacy_osph (
              config )
```

Parameters

config	`dict` A dictionary containing the script configuration.

Returns

errors: `int` The number of encountered errors.

7.4 pycompare Namespace Reference

Script to perform output consistency tests.

Functions

- main ()
  Main execution code.
- compare_files (config)
  Perform the comparison of two files.
- compare_lines (f_line, c_line, config, line_num=0, num_len=4, log_file=None)
  Perform the comparison of two file lines.
- mismatch_severities (str_f_values, str_c_values, config)
  Determine the severity of a numerical mismatch.
- parse_arguments ()
  Parse the command line arguments.
- **print_help** ()
  Print a command-line help summary.
- reformat_log (config, errors, warnings, noisy)
  Add summary information to the HTML log file.

7.4.1 Detailed Description

Script to calculate the execution time of logged operations.

Comparing the numeric output can be rendered hard by the amount of information contained in a typical output file and the necessity to determine whether a difference is actually significant or just caused by numeric noise hitting negligible values. The task of `pycompare.py` is to compare two output files, in the assumption that they were written by the FORTRAN and the C++ versions of the code and to flag all the possible inconsistencies according to various severity levels (namely: NOISE, WARNING, and ERROR).

After execution, the script returns an exit code, which is set to 0, if no error-level inconsistencies were found, or 1 otherwise. This can be used by subsequent system calls to set up a testing suite checking whether the code is able to reproduce legacy results.

The script execution requires python3.

7.4.2 Function Documentation

7.4.2.1 compare_files()

```
pycompare.compare_files (
              config )
```

The comparison is executed as a line-by-line process. In order to process files correctly, it is required that the two input files have exactly the same format (with the exception of some number formatting subtleties that are handled by regular expressions). Therefore, the first comparison step is testing whether the input files have the same number of lines. If this condition is not met, a formatting problem is assumed and every line in the C++ output file is counted as an error. Otherwise, all the numeric values found in the result are compared for consistency within warning tolerance. Warnings and errors are issued if two values do not match by a fractional amount being, respectively, below or above the threshold for warning. A special case is the detection of suspect numeric noise. This arises on very small quantities as a consequence of differences in the level of approximation or in the hardware implementation of the numeric values, which typically has negligible impact on the overall reslults, even though it can have potentially large fractional mismatches, because it is caused by values that are close to 0.

Numeric noise is filtered by taking advantage from the fact that the output files are formatted in such a way that values with similar physical meaning are written to the same output line. If the comparison results in a large fractional mismatch on a value that is more than 5 orders of magnitude smaller than the highest order of magnitude that was read from the current row, the discrepancy is flagged as a potential noise effect.

Parameters

config	`dict` A dictionary containing the script configuration.

Returns

mismatch_count: `tuple(int, int, int)` A tuple that bundles together the numbers of detected errors, warnings and noisy values.

7.4.2.2 compare_lines()

```
pycompare.compare_lines (
              f_line,
              c_line,
              config,
              line_num = 0,
              num_len = 4,
              log_file = None )
```

This function handles the line-by-line comparison of coded result files. Depending on whether a HTML log report was requested, it also undertakes the task of formatting the HTML code to show the comparison results as highlighted entries, according to the severity degree of the mismatch.

Parameters

f_line	`string` A line extracted from the FORTRAN output file.
c_line	`string` A line extracted from the C++ output file.
config	`dict` A dictionary containing the script configuration.
line_num	`int` The number of the current line (0-indexed).
num_len	`int` The number digits to format the line number tag in the HTML log.
log_file	`file` A file where to write logging information, if required.

Returns

mismatch_count: `tuple(int, int, int)` A tuple that bundles together the numbers of detected errors, warnings and noisy values.

7.4.2.3 main()

```
pycompare.main ( )
```

`main()` is the function that handles the creation of the script configuration and the execution of the comparison. It returns an integer value corresponding to the number of detected error-level inconsistencies.

Returns

errors: `int` Number of detected error-level inconsistencies.

7.4.2.4 mismatch_severities()

```
pycompare.mismatch_severities (
              str_f_values,
              str_c_values,
              config )
```

The severity scale is currently designed with the following integer codes:

0 - the values are equal

1 - the values are subject to suspect numerical noise (green fonts)

2 - the values are different but below error threshold (blue fonts)

3 - the values differ more than error threshold (red fonts)

Parameters

str_f_values	`array(string)` The strings representing the numeric values read from the FORTRAN output file.
str_c_values	`array(string)` The strings representing the numeric values read from the C++ output file.
config	`dict` A dictionary containing the configuration options from which to read the warning and the error threshold.

Returns

result: `array(int)` An array of severity codes ordered as the input numeric values.

7.4.2.5 parse_arguments()

```
pycompare.parse_arguments ( )
```

The script behaviour can be modified through a set of mandatory and optional arguments. Mandatory arguments are those required to execute a meaningful comparison and they are limited to the names of the files that need to be compared. The other arguments affect whether the script should produce an HTML log file and what level of detail needs to be included in this log.

Returns

config: `dict` A dictionary containing the script configuration.

7.4.2.6 reformat_log()

```
pycompare.reformat_log (
              config,
              errors,
              warnings,
              noisy )
```

In the case when a HTML log is requested, it is useful to obtain an overview of the detected inconsistencies. This function undertakes the task of adding a summary of the error, warning and noise counts on top of the log.

Parameters

config	`dict` A dictionary containing the script configuration.
errors	`int` The number of errors detected by the comparison.
warnings	`int` The number of warnings detected by the comparison.
noisy	`int` The number of noisy values detected by the comparison.

7.5 pydynrange Namespace Reference

Script to calculate the dynamic range of a complex matrix.

Classes

- class PlotData

  Class to represent the dynamic range memory structure.

Functions

- main ()

  Main execution code.
- get_dynamic_range (config)

  Compute the dynamic range of a matrix.
- parse_arguments ()

  Parse the command line arguments.
- **print_help** ()

  Print a command-line help summary.

7.5.1 Detailed Description

The script execution requires python3.

7.5.2 Function Documentation

7.5.2.1 get_dynamic_range()

```
pydynrange.get_dynamic_range (
              config )
```

Parameters

config	`dict` A dictionary containing the script configuration.

Returns

result: `int` The exit code of the operation (0 for success).

7.5.2.2 main()

```
pydynrange.main ( )
```

`main()` is the function that handles the creation of the script configuration and the execution of the calculation. It returns 0 on successful completion.

Returns

exit_code: `int` 0 on successful completion.

7.5.2.3 parse_arguments()

```
pydynrange.parse_arguments ( )
```

The script behaviour can be modified through a set of mandatory and optional arguments. The only mandatory argument is the name of the log file to be parsed. Additional optional arguments are an operation filter, which should be the starting sequence of the log strings to pe included in the timing calculation and the number of threads used during code execution.

Returns

config: `dict` A dictionary containing the script configuration.

7.6 pywiscombe Namespace Reference

Script to obtain a quick guess of the correct orders for a model.

Functions

- main ()

  Main execution code.
- parse_arguments ()

  Parse the command line arguments.
- **print_help** ()

  Print a command-line help summary.

7.6.1 Detailed Description

This script uses the Wiscombe criterion and its modifications to suggest the proper multipole truncation orders for a particle. It can be used to obtain an estimate of the proper internal order LI (used with the model components or with the model in far field approximation) and the proper external order LE (used for the model in near field approximation).

The script execution requires python3.

7.6.2 Function Documentation

7.6.2.1 main()

```
pywiscombe.main ( )
```

`main()` is the function that handles the creation of the script configuration and the execution of the comparison. It returns an integer value corresponding to the number of detected error-level inconsistencies.

Returns

errors: `int` Number of runtime errors (0 means successful run).

7.6.2.2 parse_arguments()

```
pywiscombe.parse_arguments ( )
```

The script behaviour can be modified through a set of mandatory and optional arguments. Mandatory arguments are those required to execute a meaningful parsing and they are limited to the names of the files that need to be read and written. The other arguments affect the format and the data fields that are sought for.

Returns

config: `dict` A dictionary containing the script configuration.

7.7 scale_model Namespace Reference

Script to re-scale an existing model by multiplicative factor.

Functions

- main ()
  Main execution code.
- parse_arguments ()
  Parse the command line arguments.
- scale_legacy_edfb (config)
  Scale a model legacy ScattererConfiguration file.
- scale_legacy_geom (config)
  Scale a model legacy GeometryConfiguration file.
- **print_help** ()
  Print a command-line help summary.

7.7.1 Detailed Description

This script is designed to create scaled versions of pre-existing models, to allow for quick production of sets of particle models characterized by the same structure, but with a different size.

The script execution requires python3.

7.7.2 Function Documentation

7.7.2.1 main()

```
scale_model.main ( )
```

`main()` is the function that handles the creation of the script configuration and the execution of the comparison. It returns an integer value corresponding to the script exit code.

Returns

errors: `int` Number of runtime errors (0 means successful run).

7.7.2.2 parse_arguments()

```
scale_model.parse_arguments ( )
```

The script behaviour can be modified through a set of mandatory and optional arguments. Mandatory arguments are those required to execute a meaningful parsing and they are limited to the names of the files that need to be read and the operational mode ("edfb" for scattering model file, "geom" for geometry model file).

Returns

config: `dict` A dictionary containing the script configuration.

7.7.2.3 scale_legacy_edfb()

```
scale_model.scale_legacy_edfb (
              config )
```

Parameters

config	`dict` A dictionary containing the script configuration.

Returns

errors: `int` The number of encountered errors.

7.7.2.4 scale_legacy_geom()

```
scale_model.scale_legacy_geom (
              config )
```

Parameters

config	`dict` A dictionary containing the script configuration.

Returns

errors: `int` The number of encountered errors.

Chapter 8

Class Documentation

8.1 CCR Struct Reference

CCR data structure.

`#include <tra_subs.h>`

Public Attributes

- double **cof**
  First coefficient.
- double **cimu**
  Second coefficient.

8.1.1 Detailed Description

A structure containing geometrical asymmetry parameter normalization coefficients.

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

- np_tmcode/src/include/tra_subs.h

8.2 CIL Struct Reference

CIL data structure.

`#include <tra_subs.h>`

Public Attributes

- int **le**

  Maximum external field expansion order.
- int **nlem**

  NLEM = LE ∗ (LE + 2)
- int **nlemt**

  NLEMT = 2 ∗ NLEM.
- int **mxmpo**

  MXMPO is read from T-matrix with IS < 0 (not used here).
- int **mxim**

  2 ∗ mxmpo - 1.

8.2.1 Detailed Description

A structure containing field expansion order configuration.

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

- np_tmcode/src/include/tra_subs.h

8.3 ClusterIterationData Class Reference

A data structure representing the information used for a single scale of the CLUSTER case.

`#include <IterationData.h>`

Public Member Functions

- ClusterIterationData (GeometryConfiguration ∗gconf, ScattererConfiguration ∗sconf, const mixMPI ∗mpidata, const int device_count)

  `ClusterIterationData` default instance constructor.
- ClusterIterationData (const ClusterIterationData &rhs)

  `ClusterIterationData` copy constructor.
- ∼**ClusterIterationData** ()

  `ClusterIterationData` instance destroyer.
- int update_orders (double ∗∗rcf, int inner_order, int outer_order)

  Update field expansion orders.

Static Public Member Functions

- static long get_size (GeometryConfiguration ∗gconf, ScattererConfiguration ∗sconf)

  Compute the memory requirements of an instance.

Public Attributes

- ParticleDescriptor * **c1**

  Pointer to a ParticleDescriptor structure.
- double * **gaps**

  Vector of geometric asymmetry factors.
- double ** **tqse**

  Components of extinction contribution to radiation torque on a single sphere along k.
- dcomplex ** **tqspe**

  Components of polarized extinction contribution to radiation torque on a single sphere along k.
- double ** **tqss**

  Components of scattering contribution to radiation torque on a single sphere along k.
- dcomplex ** **tqsps**

  Components of polarized scattering contribution to radiation torque on a single sphere along k.
- double **** **zpv**

  L-dependent coefficients of the geometric asymmetry parameter.
- double ** **gapm**

  Mean geometric asymmetry parameters.
- dcomplex ** **gappm**

  Mean geometric asymmetry parameters referred to polarization plane.
- double * **argi**

  Imaginary part of the harmonic functions argument.
- double * **args**

  Argument of the harmonic functions referred to the scattering plane.
- double ** **gap**

  Geometric asymmetry parameters.
- dcomplex ** **gapp**

  Geometric asymmetry parameters referred to polarization plane.
- double ** **tqce**

  Components of extinction contribution to radiation torque on the cluster along k.
- dcomplex ** **tqcpe**

  Components of extinction contribution to radiation torque on the cluster along k referred to polarization plane.
- double ** **tqcs**

  Components of scattering contribution to radiation torque on the cluster along k.
- dcomplex ** **tqcps**

  Components of scattering contribution to radiation torque on the cluster along k referred to polarization plane.
- double * **duk**

  Variation of unitary radiation vector.
- double ** **cextlr**

  Cluster extinction cross-section components referred to scattering plane.
- double ** **cext**

  Cluster extinction cross-section components referred to meridional plane.
- double ** **cmullr**

  Cluster Mueller Transformation Matrix components referred to scattering plane.
- double ** **cmul**

  Cluster Mueller Transformation Matrix components referred to meridional plane.
- double * **gapv**

  Geometric asymmetry parameter components.
- double * **tqev**

  Radiation extinction torque components.
- double * **tqsv**

Radiation scattering torque components.

- double * **u**

  Incident unitary vector components.
- double * **us**

  Scattered unitary vector components.
- double * **un**

  Normal unitary vector components.
- double * **uns**

  Normal scattered unitary vector components.
- double * **up**

  Incident unitary vector components on polarization plane.
- double * **ups**

  Scattered unitary vector components on polarization plane.
- double * **unmp**

  Mean unitary vector components normal to polarization plane.
- double * **unsmp**

  Mean scattered unitary vector components normal to polarization plane.
- double * **upmp**

  Mean incident unitary vector components on polarization plane.
- double * **upsmp**

  Mean scattered unitary vector components on polarization plane.
- double **scan**

  Scattering angle.
- double **cfmp**

  Control parameter on incidence direction referred to meridional plane.
- double **sfmp**

  Control parameter on scattering direction referred to meridional plane.
- double **cfsp**

  Control parameter on incidence direction referred to scattering plane.
- double **sfsp**

  Control parameter on scattering direction referred to scattering plane.
- double **sqsfi**

  SQSFI = XI^-2.
- dcomplex * **am_vector**

  Vectorized scattering coefficient matrix.
- dcomplex ** **am**

  Scattering coefficient matrix.
- dcomplex **arg**

  ANNOTATION: Argument of harmonic functions.
- double **vk**

  Vacuum magnitude of wave vector.
- double **wn**

  Wave number.
- double **xip**

  ANNOTATION: Normalization scale.
- int **number_of_scales**

  Number of scales (wavelengths) to be computed.
- int **xiblock**

  Size of the block of scales handled by the current process.
- int **firstxi**

  Index of the first scale handled by the current process.

- int **lastxi**

  Index of the last scale handled by the current process.
- int **proc_device**

  ID of the GPU used by one MPI process.
- int **refinemode**

  Refinement mode selction flag.
- bool **is_first_scale**

  flag defining a first iteration
- int **maxrefiters**

  Maximum number of refinement iterations.
- double **accuracygoal**

  Required accuracy level.

8.3.1 Constructor & Destructor Documentation

8.3.1.1 ClusterIterationData() [1/2]

```
ClusterIterationData::ClusterIterationData (
            GeometryConfiguration * gconf,
            ScattererConfiguration * sconf,
            const mixMPI * mpidata,
            const int device_count )
```

Parameters

gconf	`GeometryConfiguration *` Pointer to a `GeometryConfiguration` object.
sconf	`ScattererConfiguration *` Pointer to a `ScattererConfiguration` object.
mpidata	`mixMPI *` Pointer to a `mixMPI` object.
device_count	`const int` Number of offload devices available on the system.

8.3.1.2 ClusterIterationData() [2/2]

```
ClusterIterationData::ClusterIterationData (
            const ClusterIterationData & rhs )
```

Parameters

rhs	`const ClusterIterationData &` Reference to the `ClusterIterationData` object to be copied.

8.3.2 Member Function Documentation

8.3.2.1 get_size()

```
long ClusterIterationData::get_size (
            GeometryConfiguration * gconf,
            ScattererConfiguration * sconf )  [static]
```

Parameters

gconf	`GeometryConfiguration` * Pointer to a `GeometryConfiguration` object.
sconf	`ScattererConfiguration` * Pointer to a `ScattererConfiguration` object.

Returns

result: `long` Estimated size in bytes.

8.3.2.2 update_orders()

```
int ClusterIterationData::update_orders (
            double ** rcf,
            int inner_order,
            int outer_order )
```

Parameters

rcf	`double **` Matrix of sphere fractional radii.
inner_order	`int` The new inner expansion order to be set.
outer_order	`int` The new outer expansion order to be set.

Returns

result: `int` An exit code (0 if successful).

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

- np_tmcode/src/include/IterationData.h
- np_tmcode/src/cluster/cluster.cpp

8.4 ClusterOutputInfo Class Reference

Class to collect output information for scattering from clusters.

`#include <outputs.h>`

Public Member Functions

- ClusterOutputInfo (ScattererConfiguration *sc, GeometryConfiguration *gc, const mixMPI *mpidata, int first_xi=1, int xi_length=0)

  `ClusterOutputInfo` *default instance constructor.*
- ClusterOutputInfo (const std::string &hdf5_name)

  `ClusterOutputInfo` *constructor from HDF5 input.*
- ClusterOutputInfo (const int skip_flag)

  `ClusterOutputInfo` *constructor for the dummy NULL case*

- ~**ClusterOutputInfo** ()

  `ClusterOutputInfo` instance destroyer.
- long compute_size ()

  Get the size of a `ClusterOutputInfo` instance in bytes.
- int insert (const ClusterOutputInfo &rhs)

  Insert in the current output data the data of another block.
- int write (const std::string &output, const std::string &format)

  Write the output to a file.

Static Public Member Functions

- static long compute_size (ScattererConfiguration *sc, GeometryConfiguration *gc, int first_xi=1, int xi_length=0)

  Estimate the size of the structure that would be built for given input.

Public Attributes

- const int & **skip_flag** = _skip_flag

  Read-only view on skip_flag.
- const int & **first_xi** = _first_xi

  Read-only view on the ID of the first scale.
- int **nsph**

  Number of spheres in the aggregate.
- int **li**

  Maximum internal field expansion order.
- int **le**

  Maximum external field expansion order.
- int **lm**

  Maximum field expansion order.
- np_int **mxndm**

  Maximum coefficient matrix dimension.
- int **inpol**

  Incident polarization flag.
- int **npnt**

  Number of points for transition layer integration.
- int **npntts**

  Number of points for non-transition layer integration.
- int **iavm**

  Flag for intensity.
- int **isam**

  Flag for reference to meridional plane.
- int **idfc**

  Flag for dielectric function definition.
- double * **vec_x_coords**

  Vector of spherical components X Cartesian coordinates.
- double * **vec_y_coords**

  Vector of spherical components Y Cartesian coordinates.
- double * **vec_z_coords**

  Vector of spherical components Z Cartesian coordinates.
- double **th**

First incident radiation azimuth angle.

- double **thstp**

  Incident radiation azimuth angle step.

- double **thlst**

  Last incident radiation azimuth angle.

- double **ths**

  First scattered radiation azimuth angle.

- double **thsstp**

  Scattered radiation azimuth angle step.

- double **thslst**

  Last scattered radiation azimuth angle.

- double **ph**

  First incident radiation elevation angle.

- double **phstp**

  Incident radiation elevation angle step.

- double **phlst**

  Last incident radiation elevation angle.

- double **phs**

  First scattered radiation elevation angle.

- double **phsstp**

  Scattered radiation elevation angle step.

- double **phslst**

  Last scattered radiation elevation angle.

- int **ndirs**

  Number of directions to be explicitly solved.

- double **exri**

  Refractive index of external medium.

- int **nxi**

  Number of scales (wavelengths)

- int **xi_block_size**

  Number of scales handled by the current process.

- int **jwtm**

  Index of the wavelength for T-matrix output.

- int * **vec_jxi**

  Vector of scale (wavelength) indices.

- short * **vec_ier**

  Vector of error severities (0 - success, 1 - HJV, 2 - DME).

- double * **vec_vk**

  Vector of vacuum wave numbers.

- double * **vec_xi**

  Vector of computed scales.

- int **configurations**

  Number of sphere configurations.

- double * **vec_sphere_sizes**

  Vector of sphere sizes (all configurations for every scale).

- dcomplex * **vec_sphere_ref_indices**

  Vector of sphere refractive indices (all configurations for every scale).

- double * **vec_sphere_scs**

  Vector of sphere scattering cross-sections (all configurations for every scale).

- double * **vec_sphere_abs**

  Vector of sphere absorption cross-sections (all configurations for every scale).

- double * **vec_sphere_exs**

  Vector of sphere extinction cross-sections (all configurations for every scale).
- double * **vec_sphere_albs**

  Vector of sphere albedos (all configurations for every scale).
- double * **vec_sphere_sqscs**

  Vector of sphere scattering cross-section to geometric section ratios (all configurations for every scale).
- double * **vec_sphere_sqabs**

  Vector of sphere absorption cross-sections to geometric section ratios (all configurations for every scale).
- double * **vec_sphere_sqexs**

  Vector of sphere extinction cross-sections to geometric section ratios (all configurations for every scale).
- dcomplex * **vec_fsas**

  Vector of sphere forward scattering amplitudes (all configurations for every scale).
- double * **vec_qschus**

  Vector of QSCHU = 4 pi IMAG(FSAS) / TOTAL_GEOM_SECTION (all configurations for every scale).
- double * **vec_pschus**

  Vector of PSCHU = 4 pi REAL(FSAS) / TOTAL_GEOM_SECTION (all configurations for every scale).
- double * **vec_s0mags**

  Vector of S0MAG = ABS(FSAS) / (4 pi k^ 3) (all configurations for every scale).
- double * **vec_cosavs**

  Vector of sphere asymmetry parameters (all configurations for every scale).
- double * **vec_raprs**

  Vector of sphere radiation pressure forces (all configurations for every scale).
- double * **vec_tqek1**

  Vector of extinction contributions to radiation torques along k for parallel linear polarization (all configurations for every scale).
- double * **vec_tqsk1**

  Vector of scattering contributions to radiation torques along k for parallel linear polarization (all configurations for every scale).
- double * **vec_tqek2**

  Vector of extinction contributions to radiation torques along k for perpendicular linear polarization (all configurations for every scale).
- double * **vec_tqsk2**

  Vector of scattering contributions to radiation torques along k for perpendicular linear polarization (all configurations for every scale).
- dcomplex * **vec_fsat**

  Vector of total forward scattering amplitudes (one for each scale).
- double * **vec_qschut**

  Vector of total QSCHU (one for each scale).
- double * **vec_pschut**

  Vector of total PSCHU (one for each scale).
- double * **vec_s0magt**

  Vector of total S0MAG (one for each scale).
- double **tgs**

  Total geometric section.
- double * **vec_scc1**

  Vector of cluster scattering cross-sections (parallel polarization).
- double * **vec_scc2**

  Vector of cluster scattering cross-sections (perpendicular polarization).
- double * **vec_abc1**

  Vector of cluster absorption cross-sections (parallel polarization).
- double * **vec_abc2**

Vector of cluster absorption cross-sections (perpendicular polarization).

- double * **vec_exc1**

  Vector of cluster extinction cross-sections (parallel polarization).
- double * **vec_exc2**

  Vector of cluster extinction cross-sections (perpendicular polarization).
- double * **vec_albedc1**

  Vector of cluster albedos (parallel polarization).
- double * **vec_albedc2**

  Vector of cluster albedos (perpendicular polarization).
- double * **vec_qscamc1**

  Vector of cluster scattering to geometric cross-section ratios (parallel polarization).
- double * **vec_qscamc2**

  Vector of cluster scattering to geometric cross-section ratios (perpendicular polarization).
- double * **vec_qabsmc1**

  Vector of cluster absorption to geometric cross-section ratios (parallel polarization).
- double * **vec_qabsmc2**

  Vector of cluster absorption to geometric cross-section ratios (perpendicular polarization).
- double * **vec_qextmc1**

  Vector of cluster extinction to geometric cross-section ratios (parallel polarization).
- double * **vec_qextmc2**

  Vector of cluster extinction to geometric cross-section ratios (perpendicular polarization).
- double * **vec_sccrt1**

  Vector of cluster-to-sum-of-spheres scattering cross-section ratios (parallel polarization).
- double * **vec_sccrt2**

  Vector of cluster-to-sum-of-spheres scattering cross-section ratios (parallel polarization).
- double * **vec_abcrt1**

  Vector of cluster-to-sum-of-spheres absorption cross-section ratios (parallel polarization).
- double * **vec_abcrt2**

  Vector of cluster-to-sum-of-spheres absorption cross-section ratios (perpendicular polarization).
- double * **vec_excrt1**

  Vector of cluster-to-sum-of-spheres extinction cross-section ratios (parallel polarization).
- double * **vec_excrt2**

  Vector of cluster-to-sum-of-spheres extinction cross-section ratios (perpendicular polarization).
- dcomplex * **vec_fsac11**

  Vector of forward scattering amplitudes for polarization parallel to incidence (one per scale).
- dcomplex * **vec_fsac21**

  Vector of forward scattering amplitudes for polarization perpendicular to incidence (one per scale).
- dcomplex * **vec_fsac22**

  Vector of forward scattering amplitudes for polarization parallel to incidence (one per scale).
- dcomplex * **vec_fsac12**

  Vector of forward scattering amplitudes for polarization perpendicular to incidence (one per scale).
- double * **vec_qschuc1**

  Vector of cluster QSCHU (parallel polarization).
- double * **vec_qschuc2**

  Vector of cluster QSCHU (perpendicular polarization).
- double * **vec_pschuc1**

  Vector of cluster PSCHU (parallel polarization).
- double * **vec_pschuc2**

  Vector of cluster PSCHU (perpendicular polarization).
- double * **vec_s0magc1**

  Vector of cluster S0MAG (parallel polarization).

- double * **vec_s0magc2**

  Vector of cluster S0MAG (perpendicular polarization).
- double * **vec_cosavc1**

  Vector of cluster asymmetry parameters (parallel polarization).
- double * **vec_cosavc2**

  Vector of cluster asymmetry parameters (perpendicular polarization).
- double * **vec_raprc1**

  Vector of cluster radiation pressure forces (parallel polarization).
- double * **vec_raprc2**

  Vector of cluster radiation pressure forces (perpendicular polarization).
- double * **vec_fkc1**

  Vector of optical forces along incidence direction [N] (parallel polarization).
- double * **vec_fkc2**

  Vector of optical forces along incidence direction [N] (perpendicular polarization).
- double * **vec_dir_tidg**

  Vector of incidence azimuth directions (one per incidence azimuth).
- double * **vec_dir_pidg**

  Vector of incidence elevation directions (one per incidence elevation).
- double * **vec_dir_tsdg**

  Vector of scattering azimuth directions (one per scattering azimuth).
- double * **vec_dir_psdg**

  Vector of scattering elevation directions (one per scattering elevation).
- double * **vec_dir_scand**

  Vector of scattering angles (one per direction).
- double * **vec_dir_cfmp**

  Control parameter for incidence plane referred to meridional plane (one per direction).
- double * **vec_dir_sfmp**

  Control parameter for scattering plane referred to meridional plane (one per direction).
- double * **vec_dir_cfsp**

  Control parameter for incidence plane referred to scattering plane (one per direction).
- double * **vec_dir_sfsp**

  Control parameter for scattering plane referred to scattering plane (one per direction).
- double * **vec_dir_un**

  Components of the unitary vector perpendicular to incidence plane (three per direction).
- double * **vec_dir_uns**

  Components of the unitary vector perpendicular to scattering plane (three per direction).
- dcomplex * **vec_dir_sas11**

  Vector of sphere differential scattering amplitude with polarization parallel to parallel incidence field.
- dcomplex * **vec_dir_sas21**

  Vector of sphere differential scattering amplitude with polarization perpendicular to the parallel incidence field.
- dcomplex * **vec_dir_sas12**

  Vector of sphere differential scattering amplitude with polarization perpendicular to perpendicular incidence field.
- dcomplex * **vec_dir_sas22**

  Vector of sphere differential scattering amplitude with polarization parallel the perpendicular incidence field.
- double * **vec_dir_muls**

  Vector of sphere Mueller transormation matrices referred to meridional plane.
- double * **vec_dir_mulslr**

  Vector of sphere Mueller transormation matrices referred to scattering plane.
- dcomplex * **vec_dir_sat11**

  Vector of sphere total differential scattering amplitude with polarization parallel to parallel incidence field.
- dcomplex * **vec_dir_sat21**

Vector of sphere total differential scattering amplitude with polarization perpendicular to the parallel incidence field.

- dcomplex * **vec_dir_sat12**

  Vector of sphere total differential scattering amplitude with polarization perpendicular to perpendicular incidence field.

- dcomplex * **vec_dir_sat22**

  Vector of sphere total differential scattering amplitude with polarization parallel the perpendicular incidence field.

- double * **vec_dir_scc1**

  Vector of cluster differential scattering cross-sections (parallel polarization).

- double * **vec_dir_scc2**

  Vector of cluster differential scattering cross-sections (perpendicular polarization).

- double * **vec_dir_abc1**

  Vector of cluster differential absorption cross-sections (parallel polarization).

- double * **vec_dir_abc2**

  Vector of cluster differential absorption cross-sections (perpendicular polarization).

- double * **vec_dir_exc1**

  Vector of cluster differential extinction cross-sections (parallel polarization).

- double * **vec_dir_exc2**

  Vector of cluster differential extinction cross-sections (perpendicular polarization).

- double * **vec_dir_albedc1**

  Vector of cluster differential albedos (parallel polarization).

- double * **vec_dir_albedc2**

  Vector of cluster differential albedos (perpendicular polarization).

- double * **vec_dir_qscc1**

  Vector of differential scattering to geometric cross-section ratios (parallel polarization).

- double * **vec_dir_qscc2**

  Vector of differential scattering to geometric cross-section ratios (perpendicular polarization).

- double * **vec_dir_qabc1**

  Vector of differential absorption to geometric cross-section ratios (parallel polarization).

- double * **vec_dir_qabc2**

  Vector of differential absorption to geometric cross-section ratios (perpendicular polarization).

- double * **vec_dir_qexc1**

  Vector of differential extinction to geometric cross-section ratios (parallel polarization).

- double * **vec_dir_qexc2**

  Vector of differential extinction to geometric cross-section ratios (perpendicular polarization).

- double * **vec_dir_sccrt1**

  Vector of differential cluster-to-total scattering cross-section ratios (parallel polarization).

- double * **vec_dir_sccrt2**

  Vector of differential cluster-to-total scattering cross-section ratios (perpendicular polarization).

- double * **vec_dir_abcrt1**

  Vector of differential cluster-to-total absorption cross-section ratios (parallel polarization).

- double * **vec_dir_abcrt2**

  Vector of differential cluster-to-total absorption cross-section ratios (perpendicular polarization).

- double * **vec_dir_excrt1**

  Vector of differential cluster-to-total extinction cross-section ratios (parallel polarization).

- double * **vec_dir_excrt2**

  Vector of differential cluster-to-total extinction cross-section ratios (perpendicular polarization).

- dcomplex * **vec_dir_fsac11**

  Vector of differential cluster forward scattering amplitude with polarization parallel to parallel incidence field (one per direction and scale).

- dcomplex * **vec_dir_fsac21**

  Vector of differential cluster forward scattering amplitude with polarization perpendicular to the parallel incidence field (one per direction and scale).

- dcomplex * **vec_dir_fsac12**

  Vector of differential cluster forward scattering amplitude with polarization perpendicular to perpendicular incidence field (one per direction and scale).
- dcomplex * **vec_dir_fsac22**

  Vector of differential cluster forward scattering amplitude with polarization parallel the perpendicular incidence field (one per direction and scale).
- dcomplex * **vec_dir_sac11**

  Vector of differential cluster scattering amplitude with polarization parallel to parallel incidence field (one per direction and scale).
- dcomplex * **vec_dir_sac21**

  Vector of differential cluster scattering amplitude with polarization perpendicular to the parallel incidence field (one per direction and scale).
- dcomplex * **vec_dir_sac12**

  Vector of differential cluster scattering amplitude with polarization perpendicular to perpendicular incidence field (one per direction and scale).
- dcomplex * **vec_dir_sac22**

  Vector of differential cluster scattering amplitude with polarization parallel the perpendicular incidence field (one per direction and scale).
- double * **vec_dir_qschuc1**

  Vector of differential cluster QSCHU (parallel polarization).
- double * **vec_dir_qschuc2**

  Vector of differential cluster QSCHU (perpendicular polarization).
- double * **vec_dir_pschuc1**

  Vector of differential cluster PSCHU (parallel polarization).
- double * **vec_dir_pschuc2**

  Vector of differential cluster PSCHU (perpendicular polarization).
- double * **vec_dir_s0magc1**

  Vector of cluster differential S0MAG (parallel polarization).
- double * **vec_dir_s0magc2**

  Vector of cluster differential S0MAG (perpendicular polarization).
- double * **vec_dir_cosavc1**

  Vector of differential cluster asymmetry parameters (parallel polarization).
- double * **vec_dir_cosavc2**

  Vector of differential cluster asymmetry parameters (perpendicular polarization).
- double * **vec_dir_raprc1**

  Vector of differential cluster radiation pressure forces (1).
- double * **vec_dir_raprc2**

  Vector of differential cluster radiation pressure forces (1).
- double * **vec_dir_flc1**

  Vector of differential radiation pressure force components along the polarization direction (parallel polarization).
- double * **vec_dir_flc2**

  Vector of differential radiation pressure force components along the polarization direction (perpendicular polarization).
- double * **vec_dir_frc1**

  Vector of differential radiation pressure force components perpendicular to the polarization direction (parallel polarization).
- double * **vec_dir_frc2**

  Vector of differential radiation pressure force components perpendicular to the polarization direction (perpendicular polarization).
- double * **vec_dir_fkc1**

  Vector of differential radiation pressure force components along the incidence direction (parallel polarization).
- double * **vec_dir_fkc2**

  Vector of differential radiation pressure force components along the incidence direction (perpendicular polarization).

- double * **vec_dir_fxc1**

  Vector of differential radiation pressure force components along the X axis (parallel polarization).
- double * **vec_dir_fxc2**

  Vector of differential radiation pressure force components along the X axis (perpendicular polarization).
- double * **vec_dir_fyc1**

  Vector of differential radiation pressure force components along the Y axis (parallel polarization).
- double * **vec_dir_fyc2**

  Vector of differential radiation pressure force components along the Y axis (perpendicular polarization).
- double * **vec_dir_fzc1**

  Vector of differential radiation pressure force components along the Z axis (parallel polarization).
- double * **vec_dir_fzc2**

  Vector of differential radiation pressure force components along the Z axis (perpendicular polarization).
- double * **vec_dir_tqelc1**

  Vector of differential extinction contribution to radiation torque components along the polarization direction (parallel polarization).
- double * **vec_dir_tqelc2**

  Vector of differential extinction contribution to radiation torque components along the polarization direction (perpendicular polarization).
- double * **vec_dir_tqerc1**

  Vector of differential extinction contribution to radiation torque components perpendicular to the polarization direction (parallel polarization).
- double * **vec_dir_tqerc2**

  Vector of differential extinction contribution to radiation torque components perpendicular to the polarization direction (perpendicular polarization).
- double * **vec_dir_tqekc1**

  Vector of differential extinction contribution to radiation torque components along the incidence direction (parallel polarization).
- double * **vec_dir_tqekc2**

  Vector of differential extinction contribution to radiation torque components along the incidence direction (perpendicular polarization).
- double * **vec_dir_tqexc1**

  Vector of differential extinction contribution to radiation torque components along the X axis (parallel polarization).
- double * **vec_dir_tqexc2**

  Vector of differential extinction contribution to radiation torque components along the X axis (perpendicular polarization).
- double * **vec_dir_tqeyc1**

  Vector of differential extinction contribution to radiation torque components along the Y axis (parallel polarization).
- double * **vec_dir_tqeyc2**

  Vector of differential extinction contribution to radiation torque components along the Y axis (perpendicular polarization).
- double * **vec_dir_tqezc1**

  Vector of differential extinction contribution to radiation torque components along the Z axis (parallel polarization).
- double * **vec_dir_tqezc2**

  Vector of differential extinction contribution to radiation torque components along the Z axis (perpendicular polarization).
- double * **vec_dir_tqslc1**

  Vector of differential scattering contribution to radiation torque components along the polarization direction (parallel polarization).
- double * **vec_dir_tqslc2**

  Vector of differential scattering contribution to radiation torque components along the polarization direction (perpendicular polarization).
- double * **vec_dir_tqsrc1**

Vector of differential scattering contribution to radiation torque components perpendicular to the polarization direction (parallel polarization).

- double * **vec_dir_tqsrc2**

  Vector of differential scattering contribution to radiation torque components perpendicular to the polarization direction (perpendicular polarization).
- double * **vec_dir_tqskc1**

  Vector of differential scattering contribution to radiation torque components along the incidence direction (parallel polarization).
- double * **vec_dir_tqskc2**

  Vector of differential scattering contribution to radiation torque components along the incidence direction (perpendicular polarization).
- double * **vec_dir_tqsxc1**

  Vector of differential scattering contribution to radiation torque components along X axis (parallel polarization).
- double * **vec_dir_tqsxc2**

  Vector of differential scattering contribution to radiation torque components along X axis (perpendicular polarization).
- double * **vec_dir_tqsyc1**

  Vector of differential scattering contribution to radiation torque components along Y axis (parallel polarization).
- double * **vec_dir_tqsyc2**

  Vector of differential scattering contribution to radiation torque components along Y axis (perpendicular polarization).
- double * **vec_dir_tqszc1**

  Vector of differential scattering contribution to radiation torque components along Z axis (parallel polarization).
- double * **vec_dir_tqszc2**

  Vector of differential scattering contribution to radiation torque components along Z axis (perpendicular polarization).
- double * **vec_dir_mulc**

  Vector of cluster Mueller transormation matrices referred to meridional plane (16 per direction per scale).
- double * **vec_dir_mulclr**

  Vector of cluster Mueller transormation matrices referred to scattering plane (16 per direction per scale).

Protected Member Functions

- int write_hdf5 (const std::string &file_name)

  Write the output to a HDF5 file.
- int write_legacy (const std::string &output)

  Write the output to a legacy text file.

Protected Attributes

- int **_skip_flag**

  Flag for skipping mpisend() and mpireceive()
- int **_num_theta**

  Number of incident azimuth calculations.
- int **_num_thetas**

  Number of scattered azimuth calculations.
- int **_num_phi**

  Number of incident elevation calculations.
- int **_num_phis**

  Number of scattered elevation calculations.
- int **_first_xi**

  ID of the first computed wavelength.

8.4.1 Detailed Description

The results of the calculation can be saved in different formats. It is therefore convenient to have a proper memory structure that allows for storing the results and flushing them in any of the permitted formats with just one operation. The purpose of the ClusterOutputInfo class is to provide a wrapper for the output of the cluster scattering solver.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 ClusterOutputInfo() [1/3]

```
ClusterOutputInfo::ClusterOutputInfo (
            ScattererConfiguration * sc,
            GeometryConfiguration * gc,
            const mixMPI * mpidata,
            int first_xi = 1,
            int xi_length = 0 )
```

Parameters

sc	ScattererConfiguration * Pointer to a ScattererConfiguration instance.
gc	GeometryConfiguration * Pointer to a GeometryConfiguration instance.
mpidata	const mixMPI* Pointer to a mixMPI instance.
first_xi	int Index of the first scale in output (optional, default is 1).
xi_length	int Number of scales tobe included in output (optional, default is 0, meaning all).

8.4.2.2 ClusterOutputInfo() [2/3]

```
ClusterOutputInfo::ClusterOutputInfo (
            const std::string & hdf5_name )
```

Parameters

hdf5_name	const string & Path to the HDF5 file to be read.

8.4.2.3 ClusterOutputInfo() [3/3]

```
ClusterOutputInfo::ClusterOutputInfo (
            const int skip_flag )
```

Parameters

skip_flag	const int must be passed as the 1 constant.

8.4.3 Member Function Documentation

8.4.3.1 compute_size() [1/2]

```
long ClusterOutputInfo::compute_size ( )
```

Returns

size: `long` Estimated instance size in bytes.

8.4.3.2 compute_size() [2/2]

```
long ClusterOutputInfo::compute_size (
            ScattererConfiguration * sc,
            GeometryConfiguration * gc,
            int first_xi = 1,
            int xi_length = 0 )  [static]
```

Parameters

sc	`ScattererConfiguration *` Pointer to a `ScattererConfiguration` instance.
gc	`GeometryConfiguration *` Pointer to a `GeometryConfiguration` instance.
first_xi	`int` Index of the first scale in output (optional, default is 1).
xi_length	`int` Number of scales tobe included in output (optional, default is all).

Returns

size: `long` Estimated instance size in bytes.

8.4.3.3 insert()

```
int ClusterOutputInfo::insert (
            const ClusterOutputInfo & rhs )
```

Parameters

rhs	`const ClusterOutputInfo &` Reference to the source data block.

Returns

result: `int` Exit code (0 if successful).

8.4.3.4 write()

```
int ClusterOutputInfo::write (
            const std::string & output,
            const std::string & format )
```

Parameters

output	`const string &` Path to the output to be written.
format	`const string &` Output format (one of LEGACY or HDF5).

Returns

result: `int` Exit code (0 if successful).

8.4.3.5 write_hdf5()

```
int ClusterOutputInfo::write_hdf5 (
            const std::string & file_name )  [protected]
```

Parameters

file_name	`const string &` Path to the output to be written.

Returns

result: `int` Exit code (0 if successful).

8.4.3.6 write_legacy()

```
int ClusterOutputInfo::write_legacy (
            const std::string & output )  [protected]
```

This function takes care of writing the output using the legacy formatted ASCII structure. If the output file does not exist, it is created. If it exists, the new content overwritten.

Parameters

output	`const string &` Path to the output to be written.

Returns

result: `int` Exit code (0 if successful).

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

- np_tmcode/src/include/outputs.h
- np_tmcode/src/libnptm/outputs.cpp

8.5 List< T >::element Struct Reference

List element connector.

`#include <List.h>`

Public Attributes

- T **value**
  Value of the list element.
- element * **p_prev**
  Pointer to the previous element in the list.

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

- np_tmcode/src/include/List.h

8.6 FileSchema Class Reference

File content descriptor.

`#include <file_io.h>`

Public Member Functions

- FileSchema (int num_rec, const std::string *rec_types, const std::string *rec_names=NULL)
  FileSchema instance constructor.
- ~**FileSchema** ()
  FileSchema instance destroyer.
- int get_record_number ()
  Get the number of records in file.
- std::string * get_record_names ()
  Get a copy of the record names.
- std::string * get_record_types ()
  Get a copy of the record types.

Protected Attributes

- int **num_records**
  Number of records conained in the file.
- std::string * **record_names**
  Array of record names.
- std::string * **record_types**
  Array of record descriptors.

8.6.1 Detailed Description

Accessing binary files requires detailed knowledge of their contents. The `FileSchema` class is intended to encapsulate this information and use it as a wrapper to control I/O operations towards different file formats. Any file can be thought of as a sequence of records, which may further contain arbitrarily complex structures. By describing the structure of records, it is possible to support virtually any type of format.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 FileSchema()

```
FileSchema::FileSchema (
            int num_rec,
            const std::string * rec_types,
            const std::string * rec_names = NULL )
```

Parameters

num_rec	`int` Number of records in the file.
rec_types	`string *` Description of the records in the file.
rec_names	`string *` Names of the records in the file.

8.6.3 Member Function Documentation

8.6.3.1 get_record_names()

```
string * FileSchema::get_record_names ( )
```

Returns

rec_types: `string *` A new vector of strings with description of records.

8.6.3.2 get_record_number()

```
int FileSchema::get_record_number ( )  [inline]
```

Returns

num_records: `int` The number of records contained in the file.

8.6.3.3 get_record_types()

```
string * FileSchema::get_record_types ( )
```

Returns

rec_names: `string *` A new vector of strings with record names.

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

- np_tmcode/src/include/file_io.h
- np_tmcode/src/libnptm/file_io.cpp

8.7 GeometryConfiguration Class Reference

A class to represent the configuration of the scattering geometry.

```
#include <Configuration.h>
```

Public Member Functions

- GeometryConfiguration (int nsph, int lm, int in_pol, int npnt, int npntts, int meridional_type, int li, int le, np_int mxndm, int iavm, double *x, double *y, double *z, double in_th_start, double in_th_step, double in_th_end, double sc_th_start, double sc_th_step, double sc_th_end, double in_ph_start, double in_ph_step, double in_ph_end, double sc_ph_start, double sc_ph_step, double sc_ph_end, int jwtm)

  Build a scattering geometry configuration structure.
- GeometryConfiguration (const GeometryConfiguration &rhs)

  Build a scattering geometry configuration structure copying it from an existing one.
- ~**GeometryConfiguration** ()

  Destroy a GeometryConfiguration instance.
- double get_sph_x (int index)

  Get the X coordinate of a sphere by its index.
- double get_sph_y (int index)

  Get the Y coordinate of a sphere by its index.
- double get_sph_z (int index)

  Get the Z coordinate of a sphere by its index.

Static Public Member Functions

- static GeometryConfiguration * from_legacy (const std::string &file_name)

  Build geometry configuration from legacy configuration input file.

Public Attributes

- const int & **number_of_spheres** = _number_of_spheres

  Read-only view on number of spherical components.
- const int & **l_max** = _l_max

  Read-only view on maximum field expansion order.
- const int & **li** = _li

  Read-only view on maximum internal field expansion order.
- const int & **le** = _le

  Read-only view on maximum external field expansion order.
- const np_int & **mxndm** = _mxndm

  Read-only view on maximum dimension of allocated matrix allowance (deprecated).
- const int & **iavm** = _iavm

  Read-only view on the intensity mode flag.
- const int & **in_pol** = _in_pol

  Read-only view on incident field polarization status (0 - linear, 1 - circular).
- const int & **npnt** = _npnt

  Read-only view on number of points for transition layer integration.
- const int & **npntts** = _npntts

  Read-only view on number of points for non-transition layer integration.
- const int & **isam** = _isam

  Read-only view on type of meridional plane definition.
- const int & **jwtm** = _jwtm

  Read-only view on scale index for T-matrix output.
- const double & **in_theta_start** = _in_theta_start

  Read-only view on incident field initial azimuth.
- const double & **in_theta_step** = _in_theta_step

Read-only view on incident field azimuth step.

- const double & **in_theta_end** = _in_theta_end

  Read-only view on incident field final azimuth.

- const double & **sc_theta_start** = _sc_theta_start

  Read-only view on scattered field initial azimuth.

- const double & **sc_theta_step** = _sc_theta_step

  Read-only view on scattered field azimuth step.

- const double & **sc_theta_end** = _sc_theta_end

  Read-only view on scattered field final azimuth.

- const double & **in_phi_start** = _in_phi_start

  Read-only view on incident field initial elevation.

- const double & **in_phi_step** = _in_phi_step

  Read-only view on incident field elevation step.

- const double & **in_phi_end** = _in_phi_end

  Read-only view on incident field final elevation.

- const double & **sc_phi_start** = _sc_phi_start

  Read-only view on scattered field initial elevation.

- const double & **sc_phi_step** = _sc_phi_step

  Read-only view on scattered field elevation step.

- const double & **sc_phi_end** = _sc_phi_end

  Read-only view on scattered field final elevation.

- const short & **refine_flag** = _refine_flag

  Read-only view on flag for matrix inversion refinement.

- const short & **dyn_order_flag** = _dyn_order_flag

  Read-only view on flag for dynamic order management.

- const double & **host_ram_gb** = _host_ram_gb

  Read-only view on host RAM in GB.

- const double & **gpu_ram_gb** = _gpu_ram_gb

  Read-only view on GPU RAM in GB.

- const double & **tolerance** = _tolerance

  Read-only view on sphere overlap tolerance.

Protected Attributes

- int **_number_of_spheres**

  Number of spherical components.

- int **_l_max**

  Maximum field expansion order.

- int **_li**

  Maximum internal field expansion order.

- int **_le**

  Maximum external field expansion order.

- np_int **_mxndm**

  Maximum dimension of allocated matrix allowance (deprecated).

- int **_iavm**

  Flag for intensity.

- int **_in_pol**

  Incident field polarization status (0 - linear, 1 - circular).

- int **_npnt**

  Number of points for transition layer integration.

- int **_npntts**
  Number of points for non-transition layer integration.
- int **_isam**
  Type of meridional plane definition.
- int **_jwtm**
  Scale index of the T-matrix output.
- double **_in_theta_start**
  Incident field initial azimuth.
- double **_in_theta_step**
  Incident field azimuth step.
- double **_in_theta_end**
  Incident field final azimuth.
- double **_sc_theta_start**
  Scattered field initial azimuth.
- double **_sc_theta_step**
  Scattered field azimuth step.
- double **_sc_theta_end**
  Scattered field final azimuth.
- double **_in_phi_start**
  Incident field initial elevation.
- double **_in_phi_step**
  Incident field elevation step.
- double **_in_phi_end**
  Incident field final elevation.
- double **_sc_phi_start**
  Scattered field initial elevation.
- double **_sc_phi_step**
  Scattered field elevation step.
- double **_sc_phi_end**
  Scattered field final elevation.
- double * **_sph_x**
  Vector of spherical components X coordinates.
- double * **_sph_y**
  Vector of spherical components Y coordinates.
- double * **_sph_z**
  Vector of spherical components Z coordinates.
- short **_refine_flag**
  Flag for matrix inversion refinement.
- short **_dyn_order_flag**
  Flag for dynamic order management.
- double **_host_ram_gb**
  Host RAM in GB.
- double **_gpu_ram_gb**
  GPU RAM in GB.
- double **_tolerance**
  Sphere overlap tolerance;.

8.7.1 Detailed Description

GeometryConfiguration is a class designed to store the necessary configuration data to describe the scattering geometry, including the distribution of the particle components, the orientation of the incident and scattered radiation fields and their polarization properties.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 GeometryConfiguration() [1/2]

```
GeometryConfiguration::GeometryConfiguration (
            int nsph,
            int lm,
            int in_pol,
            int npnt,
            int npntts,
            int meridional_type,
            int li,
            int le,
            np_int mxndm,
            int iavm,
            double * x,
            double * y,
            double * z,
            double in_th_start,
            double in_th_step,
            double in_th_end,
            double sc_th_start,
            double sc_th_step,
            double sc_th_end,
            double in_ph_start,
            double in_ph_step,
            double in_ph_end,
            double sc_ph_start,
            double sc_ph_step,
            double sc_ph_end,
            int jwtm )
```

Parameters

nsph	int Number of spheres to be used in calculation.
lm	int Maximum field angular momentum expansion order.
in_pol	int Incident field polarization status
npnt	int ANNOTATION: Number of points for non transition layer integration.
npntts	int ANNOTATION: Number of points for transition layer integration.
meridional_type	int Type of meridional plane definition (<0 for incident angles, 0 if determined by incidence and observation, =1 accross z-axis for incidence and observation, >1 across z-axis as a function of incidence angles for fixed scattering).
li	int
le	int
mxndm	int
iavm	int
x	double* Vector of spherical components X coordinates.
y	double* Vector of spherical components Y coordinates.
z	double* Vector of spherical components Z coordinates.
in_th_start	double Incident field starting azimuth angle.
in_th_step	double Incident field azimuth angle step.
in_th_end	double Incident field final azimuth angle.
sc_th_start	double Scattered field starting azimuth angle.
sc_th_step	double Scattered field azimuth angle step.

Parameters

sc_th_end	`double` Scattered field final azimuth angle.
in_ph_start	`double` Incident field starting elevation angle.
in_ph_step	`double` Incident field elevation angle step.
in_ph_end	`double` Incident field final elevation angle.
sc_ph_start	`double` Scattered field starting elevation angle.
sc_ph_step	`double` Scattered field elevation angle step.
sc_ph_end	`double` Scattered field final elevation angle.
jwtm	`int` Transition Matrix layer ID.

8.7.2.2 GeometryConfiguration() [2/2]

```
GeometryConfiguration::GeometryConfiguration (
            const GeometryConfiguration & rhs )
```

Parameters

rhs	`GeometryConfiguration` preexisting object to copy from.

8.7.3 Member Function Documentation

8.7.3.1 from_legacy()

```
GeometryConfiguration * GeometryConfiguration::from_legacy (
            const std::string & file_name )  [static]
```

To allow for consistency tests and backward compatibility, geometry configurations can be built from legacy configuration files. This function replicates the approach implemented by the FORTRAN SPH and CLU codes, but using a C++ oriented work-flow.

Parameters

file_name	`string` Name of the legacy configuration data file.

Returns

config: `GeometryConfiguration*` Pointer to object containing the configuration data.

8.7.3.2 get_sph_x()

```
double GeometryConfiguration::get_sph_x (
            int index )  [inline]
```

This is a specialized function to access the X coordinate of a sphere through its index.

Parameters

index	`int` Index of the scale to be retrieved.

Returns

scale: `double` The X coordinate of the requested sphere.

8.7.3.3 get_sph_y()

```
double GeometryConfiguration::get_sph_y (
            int index )  [inline]
```

This is a specialized function to access the Y coordinate of a sphere through its index.

Parameters

index	`int` Index of the scale to be retrieved.

Returns

scale: `double` The Y coordinate of the requested sphere.

8.7.3.4 get_sph_z()

```
double GeometryConfiguration::get_sph_z (
            int index )  [inline]
```

This is a specialized function to access the Z coordinate of a sphere through its index.

Parameters

index	`int` Index of the scale to be retrieved.

Returns

scale: `double` The Z coordinate of the requested sphere.

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

- np_tmcode/src/include/Configuration.h
- np_tmcode/src/libnptm/Configuration.cpp

8.8 HDFFile Class Reference

HDF5 I/O wrapper class.

`#include <file_io.h>`

Public Member Functions

- HDFFile (const std::string &name, unsigned int flags=H5F_ACC_EXCL, hid_t fcpl_id=H5P_DEFAULT, hid_t fapl_id=H5P_DEFAULT)

  HDFFile instance constructor.
- ~**HDFFile** ()

  HDFFile instance destroyer.
- herr_t **close** ()

  Close the current file.
- hid_t **get_file_id** ()

  Get current status.
- herr_t **get_status** ()

  Get current status.
- bool **is_open** ()

  Check whether the attached file is currently open.
- herr_t read (const std::string &dataset_name, const std::string &data_type, void *buffer, hid_t mem_space_id=H5S_ALL, hid_t file_space_id=H5S_ALL, hid_t dapl_id=H5P_DEFAULT, hid_t dxpl_id=H5P_DEFAULT)

  Read data from attached file.
- herr_t write (const std::string &dataset_name, const std::string &data_type, const void *buffer, hid_t mem_space_id=H5S_ALL, hid_t file_space_id=H5S_ALL, hid_t dapl_id=H5P_DEFAULT, hid_t dxpl_id=H5P_DEFAULT)

  Write data to attached file.

Static Public Member Functions

- static HDFFile * from_schema (FileSchema &schema, const std::string &name, unsigned int flags=H5F_ACC_EXCL, hid_t fcpl_id=H5P_DEFAULT, hid_t fapl_id=H5P_DEFAULT)

  Create an empty file from a `FileSchema` instance.

Protected Attributes

- List< hid_t > * **id_list**

  Identifier list.
- std::string **file_name**

  Name of the file.
- bool **file_open_flag**

  Flag for the open file status.
- hid_t **file_id**

  File identifier handle.
- herr_t **status**

  Return status of the last operation.

8.8.1 Detailed Description

This class manages I/O operations toward HDF5 format files.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 HDFFile()

```
HDFFile::HDFFile (
            const std::string & name,
            unsigned int flags = H5F_ACC_EXCL,
            hid_t fcpl_id = H5P_DEFAULT,
            hid_t fapl_id = H5P_DEFAULT )
```

Parameters

name	`string` Name of the file.
flags	`unsigned int` File access flags (default is `H5F_ACC_EXCL`).
fcpl_id	`hid_t` File creation property list identifier (default is `H5P_DEFAULT`).
fapl_id	`hid_t` File access property list identifier (default is `H5P_DEFAULT`).

8.8.3 Member Function Documentation

8.8.3.1 from_schema()

```
HDFFile * HDFFile::from_schema (
            FileSchema & schema,
            const std::string & name,
            unsigned int flags = H5F_ACC_EXCL,
            hid_t fcpl_id = H5P_DEFAULT,
            hid_t fapl_id = H5P_DEFAULT )  [static]
```

Parameters

schema	`FileSchema &` Reference to `FileSchema` instance.
name	`string` Name of the file.
flags	`unsigned int` File access flags (default is `H5F_ACC_EXCL`).
fcpl_id	`hid_t` File creation property list identifier (default is `H5P_DEFAULT`).
fapl_id	`hid_t` File access property list identifier (default is `H5P_DEFAULT`).

Returns

hdf_file: `HDFFile *` Pointer to a new, open HDF5 file.

8.8.3.2 read()

```
herr_t HDFFile::read (
            const std::string & dataset_name,
            const std::string & data_type,
            void * buffer,
            hid_t mem_space_id = H5S_ALL,
            hid_t file_space_id = H5S_ALL,
            hid_t dapl_id = H5P_DEFAULT,
            hid_t dxpl_id = H5P_DEFAULT )
```

Parameters

dataset_name	`string` Name of the dataset to read from.
data_type	`string` Memory data type identifier.
buffer	`hid_t` Starting address of the memory sector to store the data.

Parameters

mem_space_id	`hid_t` Memory data space identifier (defaults to `H5S_ALL`).
file_space_id	`hid_t` File space identifier (defaults to `H5S_ALL`).
dapl_id	`hid_t` Data access property list identifier (defaults to `H5P_DEFAULT`).
dxpl_id	`hid_t` Data transfer property list identifier (defaults to `H5P_DEFAULT`).

Returns

status: `herr_t` Exit status of the operation.

8.8.3.3 write()

```
herr_t HDFFile::write (
            const std::string & dataset_name,
            const std::string & data_type,
            const void * buffer,
            hid_t mem_space_id = H5S_ALL,
            hid_t file_space_id = H5S_ALL,
            hid_t dapl_id = H5P_DEFAULT,
            hid_t dxpl_id = H5P_DEFAULT )
```

Parameters

dataset_name	`string` Name of the dataset to write to.
data_type	`string` Memory data type identifier.
buffer	`hid_t` Starting address of the memory sector to be written.
mem_space_id	`hid_t` Memory data space identifier (defaults to `H5S_ALL`).
file_space_id	`hid_t` File space identifier (defaults to `H5S_ALL`).
dapl_id	`hid_t` Data access property list identifier (defaults to `H5P_DEFAULT`).
dxpl_id	`hid_t` Data transfer property list identifier (defaults to `H5P_DEFAULT`).

Returns

status: `herr_t` Exit status of the operation.

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

- np_tmcode/src/include/file_io.h
- np_tmcode/src/libnptm/file_io.cpp

8.9 InclusionIterationData Class Reference

A data structure representing the information used for a single scale of the INCLUSION case.

`#include <IterationData.h>`

Public Member Functions

- InclusionIterationData (GeometryConfiguration *gconf, ScattererConfiguration *sconf, const mixMPI *mpidata, const int device_count)

  InclusionIterationData default instance constructor.
- InclusionIterationData (const InclusionIterationData &rhs)

  InclusionIterationData copy constructor.
- ~**InclusionIterationData** ()

  InclusionIterationData instance destroyer.
- int update_orders (double **rcf, int inner_order, int outer_order)

  Update field expansion orders.

Static Public Member Functions

- static long get_size (GeometryConfiguration *gconf, ScattererConfiguration *sconf)

  Compute the memory requirements of an instance.

Public Attributes

- int **nimd**

  External layer index.
- double **extr**

  External layer radius.
- ParticleDescriptor * **c1**

  Pointer to a ParticleDescriptor structure.
- double * **gaps**

  Vector of geometric asymmetry factors.
- double ** **tqse**

  Components of extinction contribution to radiation torque on a single sphere along k.
- dcomplex ** **tqspe**

  Components of polarized extinction contribution to radiation torque on a single sphere along k.
- double ** **tqss**

  Components of scattering contribution to radiation torque on a single sphere along k.
- dcomplex ** **tqsps**

  Components of polarized scattering contribution to radiation torque on a single sphere along k.
- double **** **zpv**

  L-dependent coefficients of the geometric asymmetry parameter.
- double ** **gapm**

  Mean geometric asymmetry parameters.
- dcomplex ** **gappm**

  Mean geometric asymmetry parameters referred to polarization plane.
- double * **argi**

  Imaginary part of the harmonic functions argument.
- double * **args**

  Argument of the harmonic functions referred to the scattering plane.
- double ** **gap**

  Geometric asymmetry parameters.
- dcomplex ** **gapp**

  Geometric asymmetry parameters referred to polarization plane.
- double ** **tqce**

Components of extinction contribution to radiation torque on the cluster along k.

- dcomplex ** **tqcpe**

  Components of extinction contribution to radiation torque on the cluster along k referred to polarization plane.

- double ** **tqcs**

  Components of scattering contribution to radiation torque on the cluster along k.

- dcomplex ** **tqcps**

  Components of scattering contribution to radiation torque on the cluster along k referred to polarization plane.

- double * **duk**

  Variation of unitary radiation vector.

- double ** **cextlr**

  Cluster extinction cross-section components referred to scattering plane.

- double ** **cext**

  Cluster extinction cross-section components referred to meridional plane.

- double ** **cmullr**

  Cluster Mueller Transformation Matrix components referred to scattering plane.

- double ** **cmul**

  Cluster Mueller Transformation Matrix components referred to meridional plane.

- double * **gapv**

  Geometric asymmetry parameter components.

- double * **tqev**

  Radiation extinction torque components.

- double * **tqsv**

  Radiation scattering torque components.

- double * **u**

  Incident unitary vector components.

- double * **us**

  Scattered unitary vector components.

- double * **un**

  Normal unitary vector components.

- double * **uns**

  Normal scattered unitary vector components.

- double * **up**

  Incident unitary vector components on polarization plane.

- double * **ups**

  Scattered unitary vector components on polarization plane.

- double * **unmp**

  Mean unitary vector components normal to polarization plane.

- double * **unsmp**

  Mean scattered unitary vector components normal to polarization plane.

- double * **upmp**

  Mean incident unitary vector components on polarization plane.

- double * **upsmp**

  Mean scattered unitary vector components on polarization plane.

- double **scan**

  Scattering angle.

- double **cfmp**

  Control parameter on incidence direction referred to meridional plane.

- double **sfmp**

  Control parameter on scattering direction referred to meridional plane.

- double **cfsp**

  Control parameter on incidence direction referred to scattering plane.

- double **sfsp**

  Control parameter on scattering direction referred to scattering plane.
- double **sqsfi**

  SQSFI = XI^-2.
- dcomplex * **am_vector**

  Vectorized scattering coefficient matrix.
- dcomplex ** **am**

  Scattering coefficient matrix.
- dcomplex **arg**

  Argument of harmonic functions.
- double **vk**

  Vacuum magnitude of wave vector.
- double **wn**

  Wave number.
- double **xip**

  ANNOTATION: Normalization scale.
- int **number_of_scales**

  Number of scales (wavelengths) to be computed.
- int **xiblock**

  Size of the block of scales handled by the current process.
- int **firstxi**

  Index of the first scale handled by the current process.
- int **lastxi**

  Index of the last scale handled by the current process.
- int **proc_device**

  ID of the GPU used by one MPI process.
- int **refinemode**

  Refinement mode selction flag.
- bool **is_first_scale**

  flag defining a first iteration
- int **maxrefiters**

  Maximum number of refinement iterations.
- double **accuracygoal**

  Required accuracy level.

Protected Attributes

- double * **vec_zpv**

  Vectorized geometric asymmetry parameter components.

8.9.1 Constructor & Destructor Documentation

8.9.1.1 InclusionIterationData() [1/2]

```
InclusionIterationData::InclusionIterationData (
            GeometryConfiguration * gconf,
            ScattererConfiguration * sconf,
            const mixMPI * mpidata,
            const int device_count )
```

Parameters

gconf	`GeometryConfiguration *` Pointer to a `GeometryConfiguration` object.
sconf	`ScattererConfiguration *` Pointer to a `ScattererConfiguration` object.
mpidata	`mixMPI *` Pointer to a `mixMPI` object.
device_count	`const int` Number of offload devices available on the system.

8.9.1.2 InclusionIterationData() [2/2]

```
InclusionIterationData::InclusionIterationData (
            const InclusionIterationData & rhs )
```

Parameters

rhs	`const InclusionIterationData &` Reference to the `InclusionIterationData` object to be copied.

8.9.2 Member Function Documentation

8.9.2.1 get_size()

```
long InclusionIterationData::get_size (
            GeometryConfiguration * gconf,
            ScattererConfiguration * sconf )  [static]
```

Parameters

gconf	`GeometryConfiguration *` Pointer to a `GeometryConfiguration` object.
sconf	`ScattererConfiguration *` Pointer to a `ScattererConfiguration` object.

Returns

result: `long` Estimated size in bytes.

8.9.2.2 update_orders()

```
int InclusionIterationData::update_orders (
            double ** rcf,
            int inner_order,
            int outer_order )
```

Parameters

rcf	`double **` Matrix of sphere fractional radii.
inner_order	`int` The new inner expansion order to be set.
outer_order	`int` The new outer expansion order to be set.

Returns

result: `int` An exit code (0 if successful).

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

- np_tmcode/src/include/IterationData.h
- np_tmcode/src/inclusion/inclusion.cpp

8.10 InclusionOutputInfo Class Reference

Class to collect output information for scattering from particle with inclusions.

`#include <outputs.h>`

Public Member Functions

- InclusionOutputInfo (ScattererConfiguration *sc, GeometryConfiguration *gc, const mixMPI *mpidata, int first_xi=1, int xi_length=0)

  `InclusionOutputInfo` default instance constructor.
- InclusionOutputInfo (const std::string &hdf5_name)

  `InclusionOutputInfo` constructor from HDF5 input.
- InclusionOutputInfo (const int skip_flag)

  `InclusionOutputInfo` constructor for the dummy NULL case
- ~**InclusionOutputInfo** ()

  `InclusionOutputInfo` instance destroyer.
- long compute_size ()

  Get the size of an `InclusionOutputInfo` instance in bytes.
- int insert (const InclusionOutputInfo &rhs)

  Insert in the current output data the data of another block.
- int write (const std::string &output, const std::string &format)

  Write the output to a file.

Static Public Member Functions

- static long compute_size (ScattererConfiguration *sc, GeometryConfiguration *gc, int first_xi=1, int xi_↩ length=0)

  Estimate the size of the structure that would be built for given input.

Public Attributes

- const int & **skip_flag** = _skip_flag

  Read-only view on skip_flag.
- const int & **first_xi** = _first_xi

  Read-only view on the ID of the first scale.
- int **nsph**

  Number of spheres in the aggregate.
- int **li**

  Maximum internal field expansion order.
- int **le**

  Maximum external field expansion order.
- int **lm**

  Maximum field expansion order.
- np_int **mxndm**

  Maximum coefficient matrix dimension.
- int **inpol**

  Incident polarization flag.
- int **npnt**

  Number of points for transition layer integration.
- int **npntts**

  Number of points for non-transition layer integration.
- int **iavm**

  Flag for intensity.
- int **isam**

  Flag for reference to meridional plane.
- int **idfc**

  Flag for dielectric function definition.
- double * **vec_x_coords**

  Vector of spherical components X Cartesian coordinates.
- double * **vec_y_coords**

  Vector of spherical components Y Cartesian coordinates.
- double * **vec_z_coords**

  Vector of spherical components Z Cartesian coordinates.
- double **th**

  First incident radiation azimuth angle.
- double **thstp**

  Incident radiation azimuth angle step.
- double **thlst**

  Last incident radiation azimuth angle.
- double **ths**

  First scattered radiation azimuth angle.
- double **thsstp**

  Scattered radiation azimuth angle step.
- double **thslst**

  Last scattered radiation azimuth angle.
- double **ph**

  First incident radiation elevation angle.
- double **phstp**

  Incident radiation elevation angle step.
- double **phlst**

Last incident radiation elevation angle.

- double **phs**

  First scattered radiation elevation angle.

- double **phsstp**

  Scattered radiation elevation angle step.

- double **phslst**

  Last scattered radiation elevation angle.

- int **ndirs**

  Number of directions to be explicitly solved.

- double **exri**

  Refractive index of external medium.

- int **nxi**

  Number of scales (wavelengths)

- int **xi_block_size**

  Number of scales handled by the current process.

- int **jwtm**

  Index of the wavelength for T-matrix output.

- int * **vec_jxi**

  Vector of scale (wavelength) indices.

- short * **vec_ier**

  Vector of error severities (0 - success, 1 - INDME, 2 - OSPV).

- double * **vec_vk**

  Vector of vacuum wave numbers.

- double * **vec_xi**

  Vector of computed scales.

- int **configurations**

  Number of sphere configurations.

- double * **vec_sphere_sizes**

  Vector of sphere sizes (all configurations for every scale).

- dcomplex * **vec_sphere_ref_indices**

  Vector of sphere refractive indices (all configurations for every scale).

- double * **vec_scs1**

  Vector of particle scattering cross-sections (parallel polarization).

- double * **vec_scs2**

  Vector of particle scattering cross-sections (perpendicular polarization).

- double * **vec_abs1**

  Vector of particle absorption cross-sections (parallel polarization).

- double * **vec_abs2**

  Vector of particle absorption cross-sections (perpendicular polarization).

- double * **vec_exs1**

  Vector of particle extinction cross-sections (parallel polarization).

- double * **vec_exs2**

  Vector of particle extinction cross-sections (perpendicular polarization).

- double * **vec_albeds1**

  Vector of particle albedos (parallel polarization).

- double * **vec_albeds2**

  Vector of particle albedos (perpendicular polarization).

- double * **vec_scsrt1**

  Vector of particle scattering-to-geometric cross-sections (parallel polarization).

- double * **vec_scsrt2**

  Vector of particle scattering-to-geometric cross-sections (perpendicular polarization).

- double * **vec_absrt1**

  Vector of particle absorption-to-geometric cross-sections (parallel polarization).
- double * **vec_absrt2**

  Vector of particle absorption-to-geometric cross-sections (perpendicular polarization).
- double * **vec_exsrt1**

  Vector of particle extinction-to-geometric cross-sections (parallel polarization).
- double * **vec_exsrt2**

  Vector of particle extinction-to-geometric cross-sections (perpendicular polarization).
- double * **vec_qschu1**

  Vector of particle QSCHU (parallel polarization).
- double * **vec_qschu2**

  Vector of particle QSCHU (perpendicular polarization).
- double * **vec_pschu1**

  Vector of particle PSCHU (parallel polarization).
- double * **vec_pschu2**

  Vector of particle PSCHU (perpendicular polarization).
- double * **vec_s0mag1**

  Vector of particle S0MAG (parallel polarization).
- double * **vec_s0mag2**

  Vector of particle S0MAG (perpendicular polarization).
- double * **vec_cosav1**

  Vector of particle average asymmetry parameter (parallel polarization).
- double * **vec_cosav2**

  Vector of particle average asymmetry parameter (perpendicular polarization).
- double * **vec_raprs1**

  Vector of particle average radiation pressure force (N - parallel polarization).
- double * **vec_raprs2**

  Vector of particle average radiation pressure force (N - perpendicular polarization).
- double * **vec_fk1**

  Vector of particle average radiation force along incidence direction (N - parallel polarization).
- double * **vec_fk2**

  Vector of particle average radiation force along incidence direction (N - perpendicular polarization).
- dcomplex * **vec_fsas11**

  Vector of forward scattering amplitudes for polarization parallel to incidence (one per scale).
- dcomplex * **vec_fsas21**

  Vector of forward scattering amplitudes for polarization perpendicular to incidence (one per scale).
- dcomplex * **vec_fsas22**

  Vector of forward scattering amplitudes for polarization parallel to incidence (one per scale).
- dcomplex * **vec_fsas12**

  Vector of forward scattering amplitudes for polarization perpendicular to incidence (one per scale).
- double * **vec_dir_tidg**

  Vector of incidence azimuth directions (one per incidence azimuth).
- double * **vec_dir_pidg**

  Vector of incidence elevation directions (one per incidence elevation).
- double * **vec_dir_tsdg**

  Vector of scattering azimuth directions (one per scattering azimuth).
- double * **vec_dir_psdg**

  Vector of scattering elevation directions (one per scattering elevation).
- double * **vec_dir_scand**

  Vector of scattering angles (one per direction).
- double * **vec_dir_cfmp**

Control parameter for incidence plane referred to meridional plane (one per direction).

- double * **vec_dir_sfmp**

  Control parameter for scattering plane referred to meridional plane (one per direction).
- double * **vec_dir_cfsp**

  Control parameter for incidence plane referred to scattering plane (one per direction).
- double * **vec_dir_sfsp**

  Control parameter for scattering plane referred to scattering plane (one per direction).
- double * **vec_dir_un**

  Components of the unitary vector perpendicular to incidence plane (three per direction).
- double * **vec_dir_uns**

  Components of the unitary vector perpendicular to scattering plane (three per direction).
- double * **vec_dir_scs1**

  Vector of particle differential scattering cross-sections (parallel polarization).
- double * **vec_dir_scs2**

  Vector of particle differential scattering cross-sections (perpendicular polarization).
- double * **vec_dir_abs1**

  Vector of particle differential absorption cross-sections (parallel polarization).
- double * **vec_dir_abs2**

  Vector of particle differential absorption cross-sections (perpendicular polarization).
- double * **vec_dir_exs1**

  Vector of particle differential extinction cross-sections (parallel polarization).
- double * **vec_dir_exs2**

  Vector of particle differential extinction cross-sections (perpendicular polarization).
- double * **vec_dir_albeds1**

  Vector of particle differential albedos (parallel polarization).
- double * **vec_dir_albeds2**

  Vector of particle differential albedos (perpendicular polarization).
- double * **vec_dir_scsrt1**

  Vector of particle differential scattering-to-geometric cross-sections (parallel polarization).
- double * **vec_dir_scsrt2**

  Vector of particle differential scattering-to-geometric cross-sections (perpendicular polarization).
- double * **vec_dir_absrt1**

  Vector of particle differential absorption-to-geometric cross-sections (parallel polarization).
- double * **vec_dir_absrt2**

  Vector of particle differential absorption-to-geometric cross-sections (perpendicular polarization).
- double * **vec_dir_exsrt1**

  Vector of particle differential extinction-to-geometric cross-sections (parallel polarization).
- double * **vec_dir_exsrt2**

  Vector of particle differential extinction-to-geometric cross-sections (perpendicular polarization).
- dcomplex * **vec_dir_fsas11**

  Vector of particle differential forward scattering amplitude with polarization parallel to parallel incidence field.
- dcomplex * **vec_dir_fsas21**

  Vector of particle differential forward scattering amplitude with polarization perpendicular to the parallel incidence field.
- dcomplex * **vec_dir_fsas12**

  Vector of particle differential forward scattering amplitude with polarization perpendicular to perpendicular incidence field.
- dcomplex * **vec_dir_fsas22**

  Vector of particle differential forward scattering amplitude with polarization parallel the perpendicular incidence field.
- dcomplex * **vec_dir_sas11**

  Vector of particle differential scattering amplitude with polarization parallel to parallel incidence field.

- dcomplex * **vec_dir_sas21**

  Vector of particle differential scattering amplitude with polarization perpendicular to the parallel incidence field.
- dcomplex * **vec_dir_sas12**

  Vector of particle differential scattering amplitude with polarization perpendicular to perpendicular incidence field.
- dcomplex * **vec_dir_sas22**

  Vector of particle differential scattering amplitude with polarization parallel the perpendicular incidence field.
- double * **vec_dir_qschu1**

  Vector of differential particle QSCHU (parallel polarization).
- double * **vec_dir_qschu2**

  Vector of differential particle QSCHU (perpendicular polarization).
- double * **vec_dir_pschu1**

  Vector of differential particle PSCHU (parallel polarization).
- double * **vec_dir_pschu2**

  Vector of differential particle PSCHU (perpendicular polarization).
- double * **vec_dir_s0mag1**

  Vector of particle differential S0MAG (parallel polarization).
- double * **vec_dir_s0mag2**

  Vector of particle differential S0MAG (perpendicular polarization).
- double * **vec_dir_cosav1**

  Vector of differential particle asymmetry parameters (parallel polarization).
- double * **vec_dir_cosav2**

  Vector of differential particle asymmetry parameters (perpendicular polarization).
- double * **vec_dir_rapr1**

  Vector of differential particle radiation pressure forces (1).
- double * **vec_dir_rapr2**

  Vector of differential particle radiation pressure forces (1).
- double * **vec_dir_fl1**

  Vector of differential radiation pressure force components along the polarization direction (parallel polarization).
- double * **vec_dir_fl2**

  Vector of differential radiation pressure force components along the polarization direction (perpendicular polarization).
- double * **vec_dir_fr1**

  Vector of differential radiation pressure force components perpendicular to the polarization direction (parallel polarization).
- double * **vec_dir_fr2**

  Vector of differential radiation pressure force components perpendicular to the polarization direction (perpendicular polarization).
- double * **vec_dir_fk1**

  Vector of differential radiation pressure force components along the incidence direction (parallel polarization).
- double * **vec_dir_fk2**

  Vector of differential radiation pressure force components along the incidence direction (perpendicular polarization).
- double * **vec_dir_fx1**

  Vector of differential radiation pressure force components along the X axis (parallel polarization).
- double * **vec_dir_fx2**

  Vector of differential radiation pressure force components along the X axis (perpendicular polarization).
- double * **vec_dir_fy1**

  Vector of differential radiation pressure force components along the Y axis (parallel polarization).
- double * **vec_dir_fy2**

  Vector of differential radiation pressure force components along the Y axis (perpendicular polarization).
- double * **vec_dir_fz1**

  Vector of differential radiation pressure force components along the Z axis (parallel polarization).
- double * **vec_dir_fz2**

Vector of differential radiation pressure force components along the Z axis (perpendicular polarization).

- double * **vec_dir_tqel1**

  Vector of differential extinction contribution to radiation torque components along the polarization direction (parallel polarization).

- double * **vec_dir_tqel2**

  Vector of differential extinction contribution to radiation torque components along the polarization direction (perpendicular polarization).

- double * **vec_dir_tqer1**

  Vector of differential extinction contribution to radiation torque components perpendicular to the polarization direction (parallel polarization).

- double * **vec_dir_tqer2**

  Vector of differential extinction contribution to radiation torque components perpendicular to the polarization direction (perpendicular polarization).

- double * **vec_dir_tqek1**

  Vector of differential extinction contribution to radiation torque components along the incidence direction (parallel polarization).

- double * **vec_dir_tqek2**

  Vector of differential extinction contribution to radiation torque components along the incidence direction (perpendicular polarization).

- double * **vec_dir_tqex1**

  Vector of differential extinction contribution to radiation torque components along the X axis (parallel polarization).

- double * **vec_dir_tqex2**

  Vector of differential extinction contribution to radiation torque components along the X axis (perpendicular polarization).

- double * **vec_dir_tqey1**

  Vector of differential extinction contribution to radiation torque components along the Y axis (parallel polarization).

- double * **vec_dir_tqey2**

  Vector of differential extinction contribution to radiation torque components along the Y axis (perpendicular polarization).

- double * **vec_dir_tqez1**

  Vector of differential extinction contribution to radiation torque components along the Z axis (parallel polarization).

- double * **vec_dir_tqez2**

  Vector of differential extinction contribution to radiation torque components along the Z axis (perpendicular polarization).

- double * **vec_dir_tqsl1**

  Vector of differential scattering contribution to radiation torque components along the polarization direction (parallel polarization).

- double * **vec_dir_tqsl2**

  Vector of differential scattering contribution to radiation torque components along the polarization direction (perpendicular polarization).

- double * **vec_dir_tqsr1**

  Vector of differential scattering contribution to radiation torque components perpendicular to the polarization direction (parallel polarization).

- double * **vec_dir_tqsr2**

  Vector of differential scattering contribution to radiation torque components perpendicular to the polarization direction (perpendicular polarization).

- double * **vec_dir_tqsk1**

  Vector of differential scattering contribution to radiation torque components along the incidence direction (parallel polarization).

- double * **vec_dir_tqsk2**

  Vector of differential scattering contribution to radiation torque components along the incidence direction (perpendicular polarization).

- double * **vec_dir_tqsx1**

  Vector of differential scattering contribution to radiation torque components along X axis (parallel polarization).

- double * **vec_dir_tqsx2**
  Vector of differential scattering contribution to radiation torque components along X axis (perpendicular polarization).
- double * **vec_dir_tqsy1**
  Vector of differential scattering contribution to radiation torque components along Y axis (parallel polarization).
- double * **vec_dir_tqsy2**
  Vector of differential scattering contribution to radiation torque components along Y axis (perpendicular polarization).
- double * **vec_dir_tqsz1**
  Vector of differential scattering contribution to radiation torque components along Z axis (parallel polarization).
- double * **vec_dir_tqsz2**
  Vector of differential scattering contribution to radiation torque components along Z axis (perpendicular polarization).
- double * **vec_dir_mull**
  Vector of cluster Mueller transormation matrices referred to meridional plane (16 per direction per scale).
- double * **vec_dir_mulllr**
  Vector of cluster Mueller transormation matrices referred to scattering plane (16 per direction per scale).

Protected Member Functions

- int write_hdf5 (const std::string &file_name)
  Write the output to a HDF5 file.
- int write_legacy (const std::string &output)
  Write the output to a legacy text file.

Protected Attributes

- int **_skip_flag**
  Flag for skipping mpisend() and mpireceive()
- int **_num_theta**
  Number of incident azimuth calculations.
- int **_num_thetas**
  Number of scattered azimuth calculations.
- int **_num_phi**
  Number of incident elevation calculations.
- int **_num_phis**
  Number of scattered elevation calculations.
- int **_first_xi**
  ID of the first computed wavelength.

8.10.1 Detailed Description

The results of the calculation can be saved in different formats. It is therefore convenient to have a proper memory structure that allows for storing the results and flushing them in any of the permitted formats with just one operation. The purpose of the `InclusionOutputInfo` class is to provide a wrapper for the output of the particle with inclusions scattering solver.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 InclusionOutputInfo() [1/3]

```
InclusionOutputInfo::InclusionOutputInfo (
            ScattererConfiguration * sc,
            GeometryConfiguration * gc,
            const mixMPI * mpidata,
            int first_xi = 1,
            int xi_length = 0 )
```

Parameters

sc	`ScattererConfiguration *` Pointer to a `ScattererConfiguration` instance.
gc	`GeometryConfiguration *` Pointer to a `GeometryConfiguration` instance.
mpidata	`const mixMPI*` Pointer to a mixMPI instance.
first_xi	`int` Index of the first scale in output (optional, default is 1).
xi_length	`int` Number of scales tobe included in output (optional, default is 0, meaning all).

8.10.2.2 InclusionOutputInfo() [2/3]

```
InclusionOutputInfo::InclusionOutputInfo (
            const std::string & hdf5_name )
```

Parameters

hdf5_name	`const string &` Path to the HDF5 file to be read.

8.10.2.3 InclusionOutputInfo() [3/3]

```
InclusionOutputInfo::InclusionOutputInfo (
            const int skip_flag )
```

Parameters

skip_flag	`const int` must be passed as the `1` constant.

8.10.3 Member Function Documentation

8.10.3.1 compute_size() [1/2]

```
long InclusionOutputInfo::compute_size ( )
```

Returns

size: `long` Estimated instance size in bytes.

8.10.3.2 compute_size() [2/2]

```
long InclusionOutputInfo::compute_size (
            ScattererConfiguration * sc,
            GeometryConfiguration * gc,
            int first_xi = 1,
            int xi_length = 0 )  [static]
```

Parameters

sc	`ScattererConfiguration *` Pointer to a `ScattererConfiguration` instance.
gc	`GeometryConfiguration *` Pointer to a `GeometryConfiguration` instance.
first_xi	`int` Index of the first scale in output (optional, default is 1).
xi_length	`int` Number of scales tobe included in output (optional, default is all).

Returns

size: `long` Estimated instance size in bytes.

8.10.3.3 insert()

```
int InclusionOutputInfo::insert (
            const InclusionOutputInfo & rhs )
```

Parameters

rhs	`const InclusionOutputInfo &` Reference to the source data block.

Returns

result: `int` Exit code (0 if successful).

8.10.3.4 write()

```
int InclusionOutputInfo::write (
            const std::string & output,
            const std::string & format )
```

Parameters

output	`const string &` Path to the output to be written.
format	`const string &` Output format (one of LEGACY or HDF5).

Returns

result: `int` Exit code (0 if successful).

8.10.3.5 write_hdf5()

```
int InclusionOutputInfo::write_hdf5 (
            const std::string & file_name )  [protected]
```

Parameters

file_name	`const string &` Path to the output to be written.

Returns

result: `int` Exit code (0 if successful).

8.10.3.6 write_legacy()

```
int InclusionOutputInfo::write_legacy (
            const std::string & output )  [protected]
```

This function takes care of writing the output using the legacy formatted ASCII structure. If the output file does not exist, it is created. If it exists, the new content is overwritten.

Parameters

output	`const string &` Path to the output to be written.

Returns

result: `int` Exit code (0 if successful).

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

- np_tmcode/src/include/outputs.h
- np_tmcode/src/libnptm/outputs.cpp

8.11 List< T > Class Template Reference

A class to represent dynamic lists.

```
#include <List.h>
```

Classes

- struct element

  List element connector.

Public Member Functions

- List (int length=1)
  List constructor.
- ~**List** ()
  Destroy a List instance.
- void append (T value)
  Append an element at the end of the List.
- T get (int index)
  Get the element at given index.
- int length ()
  Get the number of elements in the List.
- void set (int index, T value)
  Set an element by index and value.
- T * to_array ()
  Convert the list to a C array.

Protected Attributes

- int **size**
  Size of the List.
- element * **current**
  Pointer to element affected by last operation.
- element * **first**
  Pointer to the first element in the List.
- element * **last**
  Pointer to the last element in the List.

8.11.1 Detailed Description

template<class T>
class List< T >

This class helps in the creation and management of dynamic lists of objects, whose size is not known in advance. List offers the advantage of saving memory, since only the necessary space will be allocated, but it has the disadvantage of creating an object which is not contiguous in memory and, therefore, is very inefficient for subsequent manipulation.

For this reason, the best use of List objects is to collect all the desired members and then, once the element number is known, to convert the List to C array, by calling `List.to_array()`. This function returns a contiguous array of type T[SIZE] that can be used for indexed access.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 List()

```
template<class T >
List< T >::List (
            int length = 1 )  [inline]
```

Use the constructor `List<T>([int length])` to create a new list with a given size. If the required size is not known in advance, it is recommended to create a List with SIZE=1 (this is the default behavior) and then to append the elements dynamically, using `List.append(ELEMENT)` (where ELEMENT needs to be a value of type T, corresponding to the class template specialization). Note that, due to the default behavior, the following calls are equivalent and they both produce an integer List with size equal to 1:

```
List<int> a = List<int>(1);
List<int> b = List<int>();
```

Parameters

length	`int` The size of the list to be constructed [OPTIONAL, default=1].

8.11.3 Member Function Documentation

8.11.3.1 append()

```
template<class T >
void List< T >::append (
            T value )  [inline]
```

To dynamically create a list whose size is not known in advance, elements can be appended in an iterative way. Note that element manipulation is much more effective in a C array than in a List object. For this reason, after the List has been created, it is strongly advised to convert it to a C array by calling the function `List.to_array()`.

Parameters

value	`T` The value of the element to be appended.

8.11.3.2 get()

```
template<class T >
T List< T >::get (
            int index )  [inline]
```

Get the element specified by the index argument. The first element has index 0 and the last one has index [size - 1].

Parameters

index	`int` The index of the element to be retrieved. 0 for first.

Returns

value `T` The value of the element at the requested position.

Exceptions

ListOutOfBoundsException	Raised if the index is out of bounds.

8.11.3.3 length()

```
template<class T >
int List< T >::length ( )  [inline]
```

Get the number of elements currently stored in the List.

Returns

size `int` The size of the List.

8.11.3.4 set()

```
template<class T >
void List< T >::set (
            int index,
            T value )  [inline]
```

Set the element at the position specified by the index to the value specified by the value argument.

Parameters

index	`int` The index of the element to be set. 0 for first.
value	`int` The value to store in the pointed element.

Exceptions

ListOutOfBoundsException	Raised if the index is out of bounds.

8.11.3.5 to_array()

```
template<class T >
T * List< T >::to_array ( )  [inline]
```

The List object is useful to dynamically manage a set of objects when the number of elements is not known in advance. However, the resulting object is not contiguosly stored in memory. As a result, access to specific elements in the middle of the list is not very effective, because the list needs to be walked every time up to the desired position. In order to avoid this, `List.to_array()` makes a conversion from List to C array, returning a contiguous object, where indexed access can be used.

Returns

array `T*` A C array of type T and size equal to the List size.

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

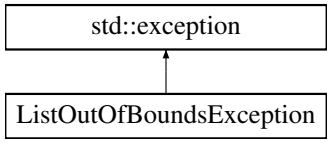
- np_tmcode/src/include/List.h

8.12 ListOutOfBoundsException Class Reference

Exception for out of bounds List requests.

`#include <errors.h>`

Inheritance diagram for ListOutOfBoundsException:



Public Member Functions

- ListOutOfBoundsException (int requested, int min, int max)

  Exception instance constructor.
- virtual const char * **what** () const throw ()

  Exception message.

Protected Attributes

- std::string **message**

  Description of the problem.

8.12.1 Constructor & Destructor Documentation

8.12.1.1 ListOutOfBoundsException()

```
ListOutOfBoundsException::ListOutOfBoundsException (
            int requested,
            int min,
            int max )  [inline]
```

Parameters

requested	`int` The index that was requested.
min	`int` The minimum index allowed by the list.
max	`int` The maximum index allowed by the list.

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

- np_tmcode/src/include/errors.h

8.13 Logger Class Reference

Logger class.

`#include <logging.h>`

Public Member Functions

- Logger (int threshold, FILE *logging_output=stdout, FILE *error_output=stderr)

  Logger instance constructor.
- ~**Logger** ()

  Logger instance destroyer.
- void err (const std::string &message)

  Print a message to the error output.
- void flush (int level=LOG_DEBG)

  Print a summary of recurrent messages and clear the stack.
- void log (const std::string &message, int level=LOG_INFO)

  Print a message, depending on its logging level.
- void push (const std::string &message)

  Push a recurrent message to the message stack.

Protected Attributes

- FILE * **err_output**

  Pointer to error stream.
- FILE * **log_output**

  Pointer to logging stream.
- std::string * **last_message**

  Last logged message.
- int **log_threshold**

  Threshold of logging level.
- long **repetitions**

  Number of identical message repetitions.

8.13.1 Detailed Description

Loggers are objects used to track the execution of a code, reporting activities such as function calls and parameter settings. They can be used to inform the user about the execution of operations at runtime (e.g. by printing messages to the terminal), as well as to record the execution history in appropriate log files. The `Logger` class offers an implementation of logging system complying with the requirements of the NP_TMcode project.

The Logger class is designed to work with open files. It is a user responsibility to check that the required log files are properly opened before use, and closed thereafter, if they are not the standard `stdout` and `stderr` streams.

8.13.2 Constructor & Destructor Documentation

8.13.2.1 Logger()

```
Logger::Logger (
            int threshold,
            FILE * logging_output = stdout,
            FILE * error_output = stderr )
```

Parameters

threshold	`int` Threshold of the messages to be included in log. Can be `LOG_DEBG` (log everything), `LOG_INFO` (give detailed information), `LOG_WARN` (log odd looking effects), or `LOG_ERRO` (print error messages, always active). The default behaviour is `LOG_WARN`.
logging_output	`FILE *` Pointer to an open output file for common messages (optional, default is `stdout`).
error_output	`FILE *` Pointer to an open output file for error messages (optional, default is `stderr`).

8.13.3 Member Function Documentation

8.13.3.1 err()

```
void Logger::err (
            const std::string & message )
```

Parameters

message	`string` The message to be printed.

8.13.3.2 flush()

```
void Logger::flush (
            int level = LOG_DEBG )
```

Parameters

level	`int` The priority level (default is `LOG_DEBG = 0`).

8.13.3.3 log()

```
void Logger::log (
            const std::string & message,
            int level = LOG_INFO )
```

Parameters

message	`string` The message to be printed.
level	`int` The priority level (default is `LOG_INFO = 1`).

8.13.3.4 push()

```
void Logger::push (
            const std::string & message )
```

When a long stream of identical messages is expected, it may be more convenient to put them in a stack. The stack is the error output stream, so that no memory is consumed. After the call stack is over, or is the message changes, the stack is flushed, meaning that a summary message is written to the logging output and the stack counter is reset to 0.

Parameters

message	`string` The message to be stacked.

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

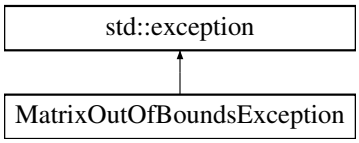
- np_tmcode/src/include/logging.h
- np_tmcode/src/libnptm/logging.cpp

8.14 MatrixOutOfBoundsException Class Reference

Exception for access requests out of matrix bounds.

`#include <errors.h>`

Inheritance diagram for MatrixOutOfBoundsException:



Public Member Functions

- MatrixOutOfBoundsException (const std::string &problem)

  Exception instance constructor.
- virtual const char * **what** () const throw ()

  Exception message.

Protected Attributes

- std::string **message**

  Description of the problem.

8.14.1 Constructor & Destructor Documentation

8.14.1.1 MatrixOutOfBoundsException()

```
MatrixOutOfBoundsException::MatrixOutOfBoundsException (
            const std::string & problem )  [inline]
```

Parameters

problem	`string` Description of the problem that occurred.

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

- np_tmcode/src/include/errors.h

8.15 mixMPI Class Reference

Structure with essential MPI data.

`#include <Commons.h>`

Public Member Functions

- **mixMPI** ()

  empty mixMPI instance constructor.
- mixMPI (const mixMPI &rhs)

  mixMPI instance constructor from an actual MPI communicator.
- ~**mixMPI** ()

  mixMPI instance destroyer.

Public Attributes

- bool **mpirunning**

  was MPI initialised?
- int **rank**

  MPI rank.
- int **nprocs**

  MPI nprocs.

8.15.1 Constructor & Destructor Documentation

8.15.1.1 mixMPI()

```
mixMPI::mixMPI (
            const mixMPI & rhs )
```

mixMPI instance constructor copying its contents from a preexisting object.

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

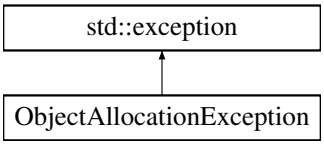
- np_tmcode/src/include/Commons.h
- np_tmcode/src/libnptm/Commons.cpp

8.16 ObjectAllocationException Class Reference

Exception for object allocation error handlers.

```
#include <errors.h>
```

Inheritance diagram for ObjectAllocationException:



Public Member Functions

- ObjectAllocationException (const std::string &name)

  Exception instance constructor.
- virtual const char * **what** () const throw ()

  Exception message.

Protected Attributes

- std::string **file_name**

  Name of the file that was accessed.

8.16.1 Constructor & Destructor Documentation

8.16.1.1 ObjectAllocationException()

```
ObjectAllocationException::ObjectAllocationException (
            const std::string & name )  [inline]
```

Parameters

name	`string` Name of the file that was accessed.

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

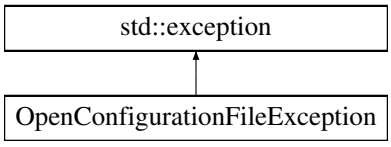
- np_tmcode/src/include/errors.h

8.17 OpenConfigurationFileException Class Reference

Exception for open file error handlers.

`#include <errors.h>`

Inheritance diagram for OpenConfigurationFileException:



Public Member Functions

- OpenConfigurationFileException (const std::string &name)

  Exception instance constructor.
- virtual const char * **what** () const throw ()

  Exception message.

Protected Attributes

- std::string **file_name**

  Name of the file that was accessed.

8.17.1 Constructor & Destructor Documentation

8.17.1.1 OpenConfigurationFileException()

```
OpenConfigurationFileException::OpenConfigurationFileException (
            const std::string & name )  [inline]
```

Parameters

name	`string` Name of the file that was accessed.

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

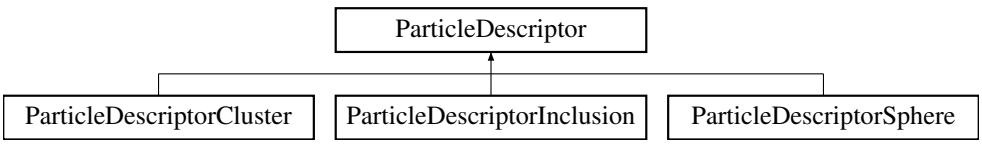
- np_tmcode/src/include/errors.h

8.18 ParticleDescriptor Class Reference

Basic data structure describing the particle model and its interaction with fields.

```
#include <Commons.h>
```

Inheritance diagram for ParticleDescriptor:



Public Member Functions

- ParticleDescriptor (GeometryConfiguration *gconf, ScattererConfiguration *sconf)

  ParticleDescriptor instance constructor.
- ParticleDescriptor (const ParticleDescriptor &rhs)

  ParticleDescriptor copy constructor.
- ~**ParticleDescriptor** ()

  ParticleDescriptor instance destroyer.
- virtual std::string get_descriptor_type ()

  Interface function to return the descriptor type as string.

Static Public Member Functions

- static long get_size (GeometryConfiguration *gconf, ScattererConfiguration *sconf)

  Compute the memory requirements of an instance.

Public Attributes

- const short & **class_type** = _class_type

  Sub-class identification code.
- const int & **nsph** = _nsph

  Read-only view of number of spheres composing the model.
- const int & **li** = _li

  Read-only view of maximum internal field expansion order.
- const int & **max_layers** = _max_layers

  Read-only view of the maximum number of layers in types.
- const int & **num_configurations** = _num_configurations

  Read-only view of number of different sphere types.
- const int & **num_layers** = _num_layers

  Read-only view of total number of layers from all sphere types.
- const int & **nhspo** = _nhspo

  Read-only view of NHSPO.
- const int & **npnt** = _npnt

  Read-only view on number of points for numerical integration in layered spheres.
- const int & **npntts** = _npntts

  Read-only view on number of points for numerical integration in transition layer.
- dcomplex ** **rmi**

  Matrix of Mie scattering b-coefficients for different orders and spheres.
- dcomplex ** **rei**

  Matrix of Matrix of Mie scattering a-coefficients for different orders and spheres.
- dcomplex ** **w**

  Matrix of multipole amplitudes for incident and scattered fields.
- dcomplex * **vint**

  Vector of intensity components.
- double * **rxx**

  Vector of sphere Cartesian X coordinates.
- double * **ryy**

  Vector of sphere Cartesian Y coordinates.
- double * **rzz**

  Vector of sphere Cartesian Z coordinates.
- double * **ros**

  Vector of sphere radii.
- double ** **rc**

  Matrix of sphere fractional transition radii.
- int * **iog**

  Vector of sphere type identifiers.
- int * **nshl**

  Vector of number of layers in sphere type.
- dcomplex * **ris**

  ANNOTATION: Square inverted coefficients.
- dcomplex * **dlri**

ANNOTATION: Refractive index variation.

- dcomplex * **dc0**

  Vector of dielectric constants.

- dcomplex * **vkt**

  Vector of complex refractive indices.

- double * **vsz**

  *Vector of sizes in units of 2*PI/LAMBDA.*

- double **gcs**

  Total geometric cross-section.

- dcomplex *** **sas**

  Tensor of sphere scattering amplitudes.

- dcomplex ** **vints**

  Array of vectors of intensity components for single spheres.

- dcomplex * **fsas**

  Vector of sphere forward scattering amplitudes.

- double * **sscs**

  Vector of scattering cross-sections for spheres.

- double * **sexs**

  Vector of extinction cross-sections for spheres.

- double * **sabs**

  Vector of absorption cross-sections for spheres.

- double * **sqscs**

  Vector of scattering efficiencies for spheres.

- double * **sqexs**

  Vector of extinction efficiencies for spheres.

- double * **sqabs**

  Vector of absorption efficiencies for spheres.

- double * **gcsv**

  Vector of geometric cross-sections for spheres.

- dcomplex * **vintt**

  Vector of field intensity components.

- dcomplex **tfsas**

  Total forward scattering amplitude of the spheres.

- dcomplex ** **tsas**

  Total scattering amplitude of the spheres.

- double **scs**

  Sphere scattering cross-section.

- double **ecs**

  Sphere extinction cross-section.

- double **acs**

  Sphere absorption cross-section.

- dcomplex ** **gis**

  ANNOTATION: Geometric tensor.

- dcomplex ** **gls**

  ANNOTATION: Geometric tensor.

- dcomplex ** **sam**

  Mean scattering amplitude components.

- const int & **le** = _le

  Read-only view of maximum external field expansion order.

- const int & **lm** = _lm

  Read-only view of maximum field expansion order.

- const int & **nlim** = _nlim

  Read-only view of NLIM.
- const int & **nlem** = _nlem

  Read-only view of NLEM.
- const int & **nlemt** = _nlemt

  Read-only view of NLEMT.
- const int & **ncou** = _ncou

  Read-only view of NCOU.
- const int & **litpo** = _litpo

  Read-only view of LITPO.
- const int & **litpos** = _litpos

  Read-only view of LITPOS.
- const int & **lmpo** = _lmpo

  Read-only view of LMPO.
- const int & **lmtpo** = _lmtpo

  Read-only view of LMTPO.
- const int & **lmtpos** = _lmtpos

  Read-only view of LMTPOS.
- const int & **nv3j** = _nv3j

  Read-only view of NV3J.
- const int & **ndi** = _ndi

  Read-only view of NDI.
- const int & **ndit** = _ndit

  Read-only view of NDIT.
- const int & **ndm** = _ndm

  Read-only view of NDM.
- dcomplex * **vh**

  ANNOTATION: Hankel vector.
- dcomplex * **vj0**

  ANNOTATION: J0 vector.
- dcomplex * **vyhj**

  ANNOTATION: Translation vector.
- dcomplex * **vyj0**

  ANNOTATION: J0 translation vector.
- dcomplex **vj**

  ANNOTATION: J vector.
- dcomplex ** **am0m**

  Transition matrix.
- dcomplex ** **fsac**

  Cluster forward scattering amplitude.
- dcomplex ** **sac**

  Cluster scattering amplitude.
- dcomplex ** **fsacm**

  Mean cluster forward scattering amplitude.
- dcomplex * **vintm**

  Mean intensity components vector.
- dcomplex * **scscp**

  Cluster polarized scattering cross-sections.
- dcomplex * **ecscp**

  Cluster polarized extinction cross-sections.
- dcomplex * **scscpm**

Mean cluster polarized scattering cross-sections.

- dcomplex * **ecscpm**

  Mean cluster polarized extinction cross-sections.

- double * **v3j0**

  ANNOTATION: J0 vector.

- double * **scsc**

  Cluster scattering cross-sections.

- double * **ecsc**

  Cluster extinction cross-sections.

- double * **scscm**

  Mean cluster scattering cross-sections.

- double * **ecscm**

  Mean cluster extinction cross-sections.

- int ** **ind3j**

  J-vector components index matrix.

- double * **rac3j**

  ANNOTATION: J-vector boundary conditions.

- dcomplex * **rm0**

  ANNOTATION: M coefficients.

- dcomplex * **re0**

  ANNOTATION: E coefficients.

- dcomplex * **rmw**

  ANNOTATION: M amplitude coefficients.

- dcomplex * **rew**

  ANNOTATION: E amplitude coefficients.

- dcomplex * **tm**

  ANNOTATION: Transition M coefficients.

- dcomplex * **te**

  ANNOTATION: Transition E coefficients.

- dcomplex * **tm0**

  ANNOTATION: Initial transition M coefficients.

- dcomplex * **te0**

  ANNOTATION: Initial transition E coefficients.

- dcomplex ** **at**

  ANNOTATION: Included particle T-matrix.

Static Public Attributes

- static const short **BASE_TYPE** = 0

  Base sub-class identification code.

- static const short **SPHERE_TYPE** = 1

  Sphere sub-class identification code.

- static const short **CLUSTER_TYPE** = 2

  Cluster sub-class identification code.

- static const short **INCLUSION_TYPE** = 3

  Inclusion sub-class identification code.

Protected Attributes

- short **_class_type**

  Sub-class identification code.
- int **_nsph**

  Number of spheres composing the model.
- int **_li**

  Maximum internal field expansion order.
- int **_max_layers**

  Maximum number of layers in known sphere types.
- int **_num_configurations**

  Number of different sphere types.
- int **_num_layers**

  Total number of layers from all sphere types.
- int **_nhspo**

  NHSPO = 2 ∗ MAX(NPNT,NPNTTS) - 1.
- int **_npnt**

  Number of points for numerical integration in layered spheres.
- int **_npntts**

  Number of points for numerical integration in transition layer.
- dcomplex ∗ **vec_rmi**

  Contiguous space for RMI.
- dcomplex ∗ **vec_rei**

  Contiguous space for REI.
- dcomplex ∗ **vec_w**

  Contiguous space for W.
- double ∗ **vec_rc**

  Contiguous space for RC.
- dcomplex ∗ **vec_sas**

  Contiguous space for SAS.
- dcomplex ∗ **vec_vints**

  Contiguous space for VINTS.
- dcomplex ∗ **vec_tsas**

  Contiguous space for TSAS.
- dcomplex ∗ **vec_gis**

  Contiguous space for GIS.
- dcomplex ∗ **vec_gls**

  Contiguous space for GLS.
- dcomplex ∗ **vec_sam**

  Contiguous space for SAM.
- int **_le**

  Maximum external field expansion order.
- int **_lm**

  Maximum field expansion order.
- int **_nlim**

  NLIM = LI ∗ (LI + 2)
- int **_nlem**

  NLEM = LE ∗ (LE + 2)
- int **_nlemt**

  NLEMT = 2 ∗ NLEM.
- int **_ncou**

NCOU = NSPH ∗ NSPH - 1.

- int **_litpo**

  LITPO = 2 ∗ LI + 1.
- int **_litpos**

  LITPOS = LITPO ∗ LITPO.
- int **_lmpo**

  LMPO = LM + 1.
- int **_lmtpo**

  LMTPO = LI + LE + 1.
- int **_lmtpos**

  LMTPOS = LMTPO ∗ LMTPO.
- int **_nv3j**

  NV3J = (LM ∗ (LM + 1) ∗ (2 ∗ LM + 7)) / 6.
- int **_ndi**

  NDI = NSPH ∗ NLIM.
- int **_ndit**

  NDIT = 2 ∗ NSPH ∗ NLIM.
- dcomplex ∗ **vec_am0m**

  Contiguous space for AM0M.
- dcomplex ∗ **vec_fsac**

  Contiguous space for FSAC.
- dcomplex ∗ **vec_sac**

  Contiguous space for SAC.
- dcomplex ∗ **vec_fsacm**

  Contiguous space for FSACM.
- int ∗ **vec_ind3j**

  Contiguous space for IND3J.
- int **_ndm**

  NDM = NDIT + NLEMT.
- dcomplex ∗ **vec_at**

  Contiguous space for AT.

8.18.1 Detailed Description

This class forms a base of the data structure collections that are used by the the code to handle different scattering problems. The class implements real members that are used by all code sections.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 ParticleDescriptor() [1/2]

```
ParticleDescriptor::ParticleDescriptor (
            GeometryConfiguration * gconf,
            ScattererConfiguration * sconf )
```

Parameters

gconf	`const GeometryConfiguration *` Pointer to GeometryConfiguration instance.
sconf	`const ScattererConfiguration *` Pointer to ScattererConfiguration instance.

8.18.2.2 ParticleDescriptor() [2/2]

```
ParticleDescriptor::ParticleDescriptor (
            const ParticleDescriptor & rhs )
```

Parameters

rhs	`const ParticleDescriptor &` Reference to ParticleDescriptor object to be copied.

8.18.3 Member Function Documentation

8.18.3.1 get_descriptor_type()

```
virtual std::string ParticleDescriptor::get_descriptor_type ( )  [inline], [virtual]
```

Returns

descriptor_type: `string` The descriptor type name.

Reimplemented in ParticleDescriptorCluster, ParticleDescriptorInclusion, and ParticleDescriptorSphere.

8.18.3.2 get_size()

```
long ParticleDescriptor::get_size (
            GeometryConfiguration * gconf,
            ScattererConfiguration * sconf )  [static]
```

Parameters

gconf	`GeometryConfiguration *` Pointer to a `GeometryConfiguration` object.
sconf	`ScattererConfiguration *` Pointer to a `ScattererConfiguration` object.

Returns

result: `long` Estimated size in bytes.

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

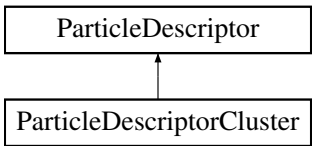
- np_tmcode/src/include/Commons.h
- np_tmcode/src/libnptm/Commons.cpp

8.19 ParticleDescriptorCluster Class Reference

The data structure describing a particle model made by a cluster of spheres.

`#include <Commons.h>`

Inheritance diagram for ParticleDescriptorCluster:



Public Member Functions

- ParticleDescriptorCluster (GeometryConfiguration *gconf, ScattererConfiguration *sconf)
  ParticleDescriptorCluster instance constructor.
- ParticleDescriptorCluster (const ParticleDescriptorCluster &rhs)
  ParticleDescriptorCluster copy constructor.
- std::string get_descriptor_type () override
  Interface function to return the descriptor type as string.
- int update_orders (int inner_order, int outer_order)
  Update the field expansion orders.

Public Member Functions inherited from ParticleDescriptor

- ParticleDescriptor (GeometryConfiguration *gconf, ScattererConfiguration *sconf)
  ParticleDescriptor instance constructor.
- ParticleDescriptor (const ParticleDescriptor &rhs)
  ParticleDescriptor copy constructor.
- ~**ParticleDescriptor** ()
  ParticleDescriptor instance destroyer.

Static Public Member Functions

- static long get_size (GeometryConfiguration *gconf, ScattererConfiguration *sconf)
  Compute the memory requirements of an instance.

Static Public Member Functions inherited from ParticleDescriptor

- static long get_size (GeometryConfiguration *gconf, ScattererConfiguration *sconf)
  Compute the memory requirements of an instance.

Additional Inherited Members

Public Attributes inherited from ParticleDescriptor

- const short & **class_type** = _class_type

  Sub-class identification code.
- const int & **nsph** = _nsph

  Read-only view of number of spheres composing the model.
- const int & **li** = _li

  Read-only view of maximum internal field expansion order.
- const int & **max_layers** = _max_layers

  Read-only view of the maximum number of layers in types.
- const int & **num_configurations** = _num_configurations

  Read-only view of number of different sphere types.
- const int & **num_layers** = _num_layers

  Read-only view of total number of layers from all sphere types.
- const int & **nhspo** = _nhspo

  Read-only view of NHSPO.
- const int & **npnt** = _npnt

  Read-only view on number of points for numerical integration in layered spheres.
- const int & **npntts** = _npntts

  Read-only view on number of points for numerical integration in transition layer.
- dcomplex ** **rmi**

  Matrix of Mie scattering b-coefficients for different orders and spheres.
- dcomplex ** **rei**

  Matrix of Matrix of Mie scattering a-coefficients for different orders and spheres.
- dcomplex ** **w**

  Matrix of multipole amplitudes for incident and scattered fields.
- dcomplex * **vint**

  Vector of intensity components.
- double * **rxx**

  Vector of sphere Cartesian X coordinates.
- double * **ryy**

  Vector of sphere Cartesian Y coordinates.
- double * **rzz**

  Vector of sphere Cartesian Z coordinates.
- double * **ros**

  Vector of sphere radii.
- double ** **rc**

  Matrix of sphere fractional transition radii.
- int * **iog**

  Vector of sphere type identifiers.
- int * **nshl**

  Vector of number of layers in sphere type.
- dcomplex * **ris**

  ANNOTATION: Square inverted coefficients.
- dcomplex * **dlri**

  ANNOTATION: Refractive index variation.
- dcomplex * **dc0**

  Vector of dielectric constants.
- dcomplex * **vkt**

Vector of complex refractive indices.

- double * **vsz**

  *Vector of sizes in units of 2*PI/LAMBDA.*

- double **gcs**

  Total geometric cross-section.

- dcomplex *** **sas**

  Tensor of sphere scattering amplitudes.

- dcomplex ** **vints**

  Array of vectors of intensity components for single spheres.

- dcomplex * **fsas**

  Vector of sphere forward scattering amplitudes.

- double * **sscs**

  Vector of scattering cross-sections for spheres.

- double * **sexs**

  Vector of extinction cross-sections for spheres.

- double * **sabs**

  Vector of absorption cross-sections for spheres.

- double * **sqscs**

  Vector of scattering efficiencies for spheres.

- double * **sqexs**

  Vector of extinction efficiencies for spheres.

- double * **sqabs**

  Vector of absorption efficiencies for spheres.

- double * **gcsv**

  Vector of geometric cross-sections for spheres.

- dcomplex * **vintt**

  Vector of field intensity components.

- dcomplex **tfsas**

  Total forward scattering amplitude of the spheres.

- dcomplex ** **tsas**

  Total scattering amplitude of the spheres.

- double **scs**

  Sphere scattering cross-section.

- double **ecs**

  Sphere extinction cross-section.

- double **acs**

  Sphere absorption cross-section.

- dcomplex ** **gis**

  ANNOTATION: Geometric tensor.

- dcomplex ** **gls**

  ANNOTATION: Geometric tensor.

- dcomplex ** **sam**

  Mean scattering amplitude components.

- const int & **le** = _le

  Read-only view of maximum external field expansion order.

- const int & **lm** = _lm

  Read-only view of maximum field expansion order.

- const int & **nlim** = _nlim

  Read-only view of NLIM.

- const int & **nlem** = _nlem

  Read-only view of NLEM.

- const int & **nlemt** = _nlemt

  Read-only view of NLEMT.
- const int & **ncou** = _ncou

  Read-only view of NCOU.
- const int & **litpo** = _litpo

  Read-only view of LITPO.
- const int & **litpos** = _litpos

  Read-only view of LITPOS.
- const int & **lmpo** = _lmpo

  Read-only view of LMPO.
- const int & **lmtpo** = _lmtpo

  Read-only view of LMTPO.
- const int & **lmtpos** = _lmtpos

  Read-only view of LMTPOS.
- const int & **nv3j** = _nv3j

  Read-only view of NV3J.
- const int & **ndi** = _ndi

  Read-only view of NDI.
- const int & **ndit** = _ndit

  Read-only view of NDIT.
- const int & **ndm** = _ndm

  Read-only view of NDM.
- dcomplex * **vh**

  ANNOTATION: Hankel vector.
- dcomplex * **vj0**

  ANNOTATION: J0 vector.
- dcomplex * **vyhj**

  ANNOTATION: Translation vector.
- dcomplex * **vyj0**

  ANNOTATION: J0 translation vector.
- dcomplex **vj**

  ANNOTATION: J vector.
- dcomplex ** **am0m**

  Transition matrix.
- dcomplex ** **fsac**

  Cluster forward scattering amplitude.
- dcomplex ** **sac**

  Cluster scattering amplitude.
- dcomplex ** **fsacm**

  Mean cluster forward scattering amplitude.
- dcomplex * **vintm**

  Mean intensity components vector.
- dcomplex * **scscp**

  Cluster polarized scattering cross-sections.
- dcomplex * **ecscp**

  Cluster polarized extinction cross-sections.
- dcomplex * **scscpm**

  Mean cluster polarized scattering cross-sections.
- dcomplex * **ecscpm**

  Mean cluster polarized extinction cross-sections.
- double * **v3j0**

ANNOTATION: J0 vector.

- double * **scsc**

  Cluster scattering cross-sections.
- double * **ecsc**

  Cluster extinction cross-sections.
- double * **scscm**

  Mean cluster scattering cross-sections.
- double * **ecscm**

  Mean cluster extinction cross-sections.
- int ** **ind3j**

  J-vector components index matrix.
- double * **rac3j**

  ANNOTATION: J-vector boundary conditions.
- dcomplex * **rm0**

  ANNOTATION: M coefficients.
- dcomplex * **re0**

  ANNOTATION: E coefficients.
- dcomplex * **rmw**

  ANNOTATION: M amplitude coefficients.
- dcomplex * **rew**

  ANNOTATION: E amplitude coefficients.
- dcomplex * **tm**

  ANNOTATION: Transition M coefficients.
- dcomplex * **te**

  ANNOTATION: Transition E coefficients.
- dcomplex * **tm0**

  ANNOTATION: Initial transition M coefficients.
- dcomplex * **te0**

  ANNOTATION: Initial transition E coefficients.
- dcomplex ** **at**

  ANNOTATION: Included particle T-matrix.

Static Public Attributes inherited from ParticleDescriptor

- static const short **BASE_TYPE** = 0

  Base sub-class identification code.
- static const short **SPHERE_TYPE** = 1

  Sphere sub-class identification code.
- static const short **CLUSTER_TYPE** = 2

  Cluster sub-class identification code.
- static const short **INCLUSION_TYPE** = 3

  Inclusion sub-class identification code.

Protected Attributes inherited from ParticleDescriptor

- short **_class_type**

  Sub-class identification code.
- int **_nsph**

  Number of spheres composing the model.
- int **_li**

  Maximum internal field expansion order.
- int **_max_layers**

  Maximum number of layers in known sphere types.
- int **_num_configurations**

  Number of different sphere types.
- int **_num_layers**

  Total number of layers from all sphere types.
- int **_nhspo**

  NHSPO = 2 ∗ MAX(NPNT,NPNTTS) - 1.
- int **_npnt**

  Number of points for numerical integration in layered spheres.
- int **_npntts**

  Number of points for numerical integration in transition layer.
- dcomplex ∗ **vec_rmi**

  Contiguous space for RMI.
- dcomplex ∗ **vec_rei**

  Contiguous space for REI.
- dcomplex ∗ **vec_w**

  Contiguous space for W.
- double ∗ **vec_rc**

  Contiguous space for RC.
- dcomplex ∗ **vec_sas**

  Contiguous space for SAS.
- dcomplex ∗ **vec_vints**

  Contiguous space for VINTS.
- dcomplex ∗ **vec_tsas**

  Contiguous space for TSAS.
- dcomplex ∗ **vec_gis**

  Contiguous space for GIS.
- dcomplex ∗ **vec_gls**

  Contiguous space for GLS.
- dcomplex ∗ **vec_sam**

  Contiguous space for SAM.
- int **_le**

  Maximum external field expansion order.
- int **_lm**

  Maximum field expansion order.
- int **_nlim**

  NLIM = LI ∗ (LI + 2)
- int **_nlem**

  NLEM = LE ∗ (LE + 2)
- int **_nlemt**

  NLEMT = 2 ∗ NLEM.
- int **_ncou**

NCOU = NSPH ∗ NSPH - 1.

- int **_litpo**

  LITPO = 2 ∗ LI + 1.
- int **_litpos**

  LITPOS = LITPO ∗ LITPO.
- int **_lmpo**

  LMPO = LM + 1.
- int **_lmtpo**

  LMTPO = LI + LE + 1.
- int **_lmtpos**

  LMTPOS = LMTPO ∗ LMTPO.
- int **_nv3j**

  NV3J = (LM ∗ (LM + 1) ∗ (2 ∗ LM + 7)) / 6.
- int **_ndi**

  NDI = NSPH ∗ NLIM.
- int **_ndit**

  NDIT = 2 ∗ NSPH ∗ NLIM.
- dcomplex ∗ **vec_am0m**

  Contiguous space for AM0M.
- dcomplex ∗ **vec_fsac**

  Contiguous space for FSAC.
- dcomplex ∗ **vec_sac**

  Contiguous space for SAC.
- dcomplex ∗ **vec_fsacm**

  Contiguous space for FSACM.
- int ∗ **vec_ind3j**

  Contiguous space for IND3J.
- int **_ndm**

  NDM = NDIT + NLEMT.
- dcomplex ∗ **vec_at**

  Contiguous space for AT.

8.19.1 Detailed Description

This class is used to solve the problem of a cluster of spherical particles. It replaces the C1 and C1_AddOns class implementation of the C1 FORTRAN common block.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 ParticleDescriptorCluster() [1/2]

```
ParticleDescriptorCluster::ParticleDescriptorCluster (
            GeometryConfiguration * gconf,
            ScattererConfiguration * sconf )
```

Parameters

gconf	`const GeometryConfiguration *` Pointer to GeometryConfiguration instance.
sconf	`const ScattererConfiguration *` Pointer to ScattererConfiguration instance.

8.19.2.2 ParticleDescriptorCluster() [2/2]

```
ParticleDescriptorCluster::ParticleDescriptorCluster (
            const ParticleDescriptorCluster & rhs )
```

Parameters

rhs	const ParticleDescriptorCluster & Reference to ParticleDescriptorCluster object to be copied.

8.19.3 Member Function Documentation

8.19.3.1 get_descriptor_type()

```
std::string ParticleDescriptorCluster::get_descriptor_type ( )  [inline], [override], [virtual]
```

Returns

descriptor_type: string The descriptor type name.

Reimplemented from ParticleDescriptor.

8.19.3.2 get_size()

```
long ParticleDescriptorCluster::get_size (
            GeometryConfiguration * gconf,
            ScattererConfiguration * sconf )  [static]
```

Parameters

gconf	GeometryConfiguration * Pointer to a GeometryConfiguration object.
sconf	ScattererConfiguration * Pointer to a ScattererConfiguration object.

Returns

result: long Estimated size in bytes.

8.19.3.3 update_orders()

```
int ParticleDescriptorCluster::update_orders (
            int inner_order,
            int outer_order )
```

Parameters

inner_order	int The new inner expansion order to be set.
outer_order	int The new outer expansion order to be set.

Returns

result: `int` An exit code (0 if successful).

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

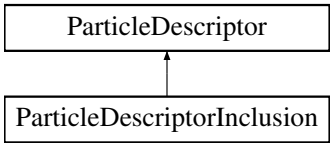
- np_tmcode/src/include/Commons.h
- np_tmcode/src/libnptm/Commons.cpp

8.20 ParticleDescriptorInclusion Class Reference

The data structure describing a particle model for a sphere with inclusions.

`#include <Commons.h>`

Inheritance diagram for ParticleDescriptorInclusion:



Public Member Functions

- ParticleDescriptorInclusion (GeometryConfiguration *gconf, ScattererConfiguration *sconf)
  ParticleDescriptorInclusion instance constructor.
- ParticleDescriptorInclusion (const ParticleDescriptorInclusion &rhs)
  ParticleDescriptorInclusion copy constructor.
- std::string get_descriptor_type () override
  Interface function to return the descriptor type as string.
- int update_orders (int inner_order, int outer_order)
  Update the field expansion orders.

Public Member Functions inherited from ParticleDescriptor

- ParticleDescriptor (GeometryConfiguration *gconf, ScattererConfiguration *sconf)
  ParticleDescriptor instance constructor.
- ParticleDescriptor (const ParticleDescriptor &rhs)
  ParticleDescriptor copy constructor.
- ~**ParticleDescriptor** ()
  ParticleDescriptor instance destroyer.

Static Public Member Functions

- static long get_size (GeometryConfiguration *gconf, ScattererConfiguration *sconf)
  Compute the memory requirements of an instance.

Static Public Member Functions inherited from ParticleDescriptor

- static long get_size (GeometryConfiguration *gconf, ScattererConfiguration *sconf)

  Compute the memory requirements of an instance.

Additional Inherited Members

Public Attributes inherited from ParticleDescriptor

- const short & **class_type** = _class_type

  Sub-class identification code.
- const int & **nsph** = _nsph

  Read-only view of number of spheres composing the model.
- const int & **li** = _li

  Read-only view of maximum internal field expansion order.
- const int & **max_layers** = _max_layers

  Read-only view of the maximum number of layers in types.
- const int & **num_configurations** = _num_configurations

  Read-only view of number of different sphere types.
- const int & **num_layers** = _num_layers

  Read-only view of total number of layers from all sphere types.
- const int & **nhspo** = _nhspo

  Read-only view of NHSPO.
- const int & **npnt** = _npnt

  Read-only view on number of points for numerical integration in layered spheres.
- const int & **npntts** = _npntts

  Read-only view on number of points for numerical integration in transition layer.
- dcomplex ** **rmi**

  Matrix of Mie scattering b-coefficients for different orders and spheres.
- dcomplex ** **rei**

  Matrix of Matrix of Mie scattering a-coefficients for different orders and spheres.
- dcomplex ** **w**

  Matrix of multipole amplitudes for incident and scattered fields.
- dcomplex * **vint**

  Vector of intensity components.
- double * **rxx**

  Vector of sphere Cartesian X coordinates.
- double * **ryy**

  Vector of sphere Cartesian Y coordinates.
- double * **rzz**

  Vector of sphere Cartesian Z coordinates.
- double * **ros**

  Vector of sphere radii.
- double ** **rc**

  Matrix of sphere fractional transition radii.
- int * **iog**

  Vector of sphere type identifiers.
- int * **nshl**

  Vector of number of layers in sphere type.
- dcomplex * **ris**

ANNOTATION: Square inverted coefficients.

- dcomplex * **dlri**

  ANNOTATION: Refractive index variation.

- dcomplex * **dc0**

  Vector of dielectric constants.

- dcomplex * **vkt**

  Vector of complex refractive indices.

- double * **vsz**

  *Vector of sizes in units of 2*PI/LAMBDA.*

- double **gcs**

  Total geometric cross-section.

- dcomplex *** **sas**

  Tensor of sphere scattering amplitudes.

- dcomplex ** **vints**

  Array of vectors of intensity components for single spheres.

- dcomplex * **fsas**

  Vector of sphere forward scattering amplitudes.

- double * **sscs**

  Vector of scattering cross-sections for spheres.

- double * **sexs**

  Vector of extinction cross-sections for spheres.

- double * **sabs**

  Vector of absorption cross-sections for spheres.

- double * **sqscs**

  Vector of scattering efficiencies for spheres.

- double * **sqexs**

  Vector of extinction efficiencies for spheres.

- double * **sqabs**

  Vector of absorption efficiencies for spheres.

- double * **gcsv**

  Vector of geometric cross-sections for spheres.

- dcomplex * **vintt**

  Vector of field intensity components.

- dcomplex **tfsas**

  Total forward scattering amplitude of the spheres.

- dcomplex ** **tsas**

  Total scattering amplitude of the spheres.

- double **scs**

  Sphere scattering cross-section.

- double **ecs**

  Sphere extinction cross-section.

- double **acs**

  Sphere absorption cross-section.

- dcomplex ** **gis**

  ANNOTATION: Geometric tensor.

- dcomplex ** **gls**

  ANNOTATION: Geometric tensor.

- dcomplex ** **sam**

  Mean scattering amplitude components.

- const int & **le** = _le

  Read-only view of maximum external field expansion order.

- const int & **lm** = _lm

  Read-only view of maximum field expansion order.
- const int & **nlim** = _nlim

  Read-only view of NLIM.
- const int & **nlem** = _nlem

  Read-only view of NLEM.
- const int & **nlemt** = _nlemt

  Read-only view of NLEMT.
- const int & **ncou** = _ncou

  Read-only view of NCOU.
- const int & **litpo** = _litpo

  Read-only view of LITPO.
- const int & **litpos** = _litpos

  Read-only view of LITPOS.
- const int & **lmpo** = _lmpo

  Read-only view of LMPO.
- const int & **lmtpo** = _lmtpo

  Read-only view of LMTPO.
- const int & **lmtpos** = _lmtpos

  Read-only view of LMTPOS.
- const int & **nv3j** = _nv3j

  Read-only view of NV3J.
- const int & **ndi** = _ndi

  Read-only view of NDI.
- const int & **ndit** = _ndit

  Read-only view of NDIT.
- const int & **ndm** = _ndm

  Read-only view of NDM.
- dcomplex * **vh**

  ANNOTATION: Hankel vector.
- dcomplex * **vj0**

  ANNOTATION: J0 vector.
- dcomplex * **vyhj**

  ANNOTATION: Translation vector.
- dcomplex * **vyj0**

  ANNOTATION: J0 translation vector.
- dcomplex **vj**

  ANNOTATION: J vector.
- dcomplex ** **am0m**

  Transition matrix.
- dcomplex ** **fsac**

  Cluster forward scattering amplitude.
- dcomplex ** **sac**

  Cluster scattering amplitude.
- dcomplex ** **fsacm**

  Mean cluster forward scattering amplitude.
- dcomplex * **vintm**

  Mean intensity components vector.
- dcomplex * **scscp**

  Cluster polarized scattering cross-sections.
- dcomplex * **ecscp**

Cluster polarized extinction cross-sections.

- dcomplex * **scscpm**

  Mean cluster polarized scattering cross-sections.

- dcomplex * **ecscpm**

  Mean cluster polarized extinction cross-sections.

- double * **v3j0**

  ANNOTATION: J0 vector.

- double * **scsc**

  Cluster scattering cross-sections.

- double * **ecsc**

  Cluster extinction cross-sections.

- double * **scscm**

  Mean cluster scattering cross-sections.

- double * **ecscm**

  Mean cluster extinction cross-sections.

- int ** **ind3j**

  J-vector components index matrix.

- double * **rac3j**

  ANNOTATION: J-vector boundary conditions.

- dcomplex * **rm0**

  ANNOTATION: M coefficients.

- dcomplex * **re0**

  ANNOTATION: E coefficients.

- dcomplex * **rmw**

  ANNOTATION: M amplitude coefficients.

- dcomplex * **rew**

  ANNOTATION: E amplitude coefficients.

- dcomplex * **tm**

  ANNOTATION: Transition M coefficients.

- dcomplex * **te**

  ANNOTATION: Transition E coefficients.

- dcomplex * **tm0**

  ANNOTATION: Initial transition M coefficients.

- dcomplex * **te0**

  ANNOTATION: Initial transition E coefficients.

- dcomplex ** **at**

  ANNOTATION: Included particle T-matrix.

Static Public Attributes inherited from ParticleDescriptor

- static const short **BASE_TYPE** = 0

  Base sub-class identification code.

- static const short **SPHERE_TYPE** = 1

  Sphere sub-class identification code.

- static const short **CLUSTER_TYPE** = 2

  Cluster sub-class identification code.

- static const short **INCLUSION_TYPE** = 3

  Inclusion sub-class identification code.

Protected Attributes inherited from ParticleDescriptor

- short **_class_type**

  Sub-class identification code.
- int **_nsph**

  Number of spheres composing the model.
- int **_li**

  Maximum internal field expansion order.
- int **_max_layers**

  Maximum number of layers in known sphere types.
- int **_num_configurations**

  Number of different sphere types.
- int **_num_layers**

  Total number of layers from all sphere types.
- int **_nhspo**

  NHSPO = 2 ∗ MAX(NPNT,NPNTTS) - 1.
- int **_npnt**

  Number of points for numerical integration in layered spheres.
- int **_npntts**

  Number of points for numerical integration in transition layer.
- dcomplex ∗ **vec_rmi**

  Contiguous space for RMI.
- dcomplex ∗ **vec_rei**

  Contiguous space for REI.
- dcomplex ∗ **vec_w**

  Contiguous space for W.
- double ∗ **vec_rc**

  Contiguous space for RC.
- dcomplex ∗ **vec_sas**

  Contiguous space for SAS.
- dcomplex ∗ **vec_vints**

  Contiguous space for VINTS.
- dcomplex ∗ **vec_tsas**

  Contiguous space for TSAS.
- dcomplex ∗ **vec_gis**

  Contiguous space for GIS.
- dcomplex ∗ **vec_gls**

  Contiguous space for GLS.
- dcomplex ∗ **vec_sam**

  Contiguous space for SAM.
- int **_le**

  Maximum external field expansion order.
- int **_lm**

  Maximum field expansion order.
- int **_nlim**

  NLIM = LI ∗ (LI + 2)
- int **_nlem**

  NLEM = LE ∗ (LE + 2)
- int **_nlemt**

  NLEMT = 2 ∗ NLEM.
- int **_ncou**

NCOU = NSPH ∗ NSPH - 1.

- int **_litpo**

  LITPO = 2 ∗ LI + 1.
- int **_litpos**

  LITPOS = LITPO ∗ LITPO.
- int **_lmpo**

  LMPO = LM + 1.
- int **_lmtpo**

  LMTPO = LI + LE + 1.
- int **_lmtpos**

  LMTPOS = LMTPO ∗ LMTPO.
- int **_nv3j**

  NV3J = (LM ∗ (LM + 1) ∗ (2 ∗ LM + 7)) / 6.
- int **_ndi**

  NDI = NSPH ∗ NLIM.
- int **_ndit**

  NDIT = 2 ∗ NSPH ∗ NLIM.
- dcomplex ∗ **vec_am0m**

  Contiguous space for AM0M.
- dcomplex ∗ **vec_fsac**

  Contiguous space for FSAC.
- dcomplex ∗ **vec_sac**

  Contiguous space for SAC.
- dcomplex ∗ **vec_fsacm**

  Contiguous space for FSACM.
- int ∗ **vec_ind3j**

  Contiguous space for IND3J.
- int **_ndm**

  NDM = NDIT + NLEMT.
- dcomplex ∗ **vec_at**

  Contiguous space for AT.

8.20.1 Detailed Description

This class is used to solve the problem of a spherical particle with a cluster of inclusions. It replaces the C1 and C1_AddOns class implementation of the C1 FORTRAN common block.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 ParticleDescriptorInclusion() [1/2]

```
ParticleDescriptorInclusion::ParticleDescriptorInclusion (
            GeometryConfiguration * gconf,
            ScattererConfiguration * sconf )
```

Parameters

gconf	`const GeometryConfiguration *` Pointer to GeometryConfiguration instance.
sconf	`const ScattererConfiguration *` Pointer to ScattererConfiguration instance.

8.20.2.2 ParticleDescriptorInclusion() [2/2]

```
ParticleDescriptorInclusion::ParticleDescriptorInclusion (
            const ParticleDescriptorInclusion & rhs )
```

Parameters

rhs	const ParticleDescriptorInclusion & Reference to ParticleDescriptorInclusion object to be copied.

8.20.3 Member Function Documentation

8.20.3.1 get_descriptor_type()

```
std::string ParticleDescriptorInclusion::get_descriptor_type ( )  [inline], [override], [virtual]
```

Returns

descriptor_type: string The descriptor type name.

Reimplemented from ParticleDescriptor.

8.20.3.2 get_size()

```
long ParticleDescriptorInclusion::get_size (
            GeometryConfiguration * gconf,
            ScattererConfiguration * sconf )  [static]
```

Parameters

gconf	GeometryConfiguration * Pointer to a GeometryConfiguration object.
sconf	ScattererConfiguration * Pointer to a ScattererConfiguration object.

Returns

result: long Estimated size in bytes.

8.20.3.3 update_orders()

```
int ParticleDescriptorInclusion::update_orders (
            int inner_order,
            int outer_order )
```

Parameters

inner_order	int The new inner expansion order to be set.
outer_order	int The new outer expansion order to be set.

Returns

result: `int` An exit code (0 if successful).

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

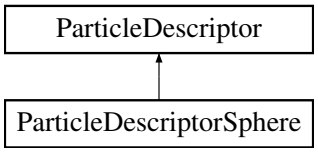
- np_tmcode/src/include/Commons.h
- np_tmcode/src/libnptm/Commons.cpp

8.21 ParticleDescriptorSphere Class Reference

The data structure describing a spherical particle model.

`#include <Commons.h>`

Inheritance diagram for ParticleDescriptorSphere:



Public Member Functions

- ParticleDescriptorSphere (GeometryConfiguration *gconf, ScattererConfiguration *sconf)
  ParticleDescriptorSphere instance constructor.
- ParticleDescriptorSphere (const ParticleDescriptorSphere &rhs)
  ParticleDescriptorSphere copy constructor.
- std::string get_descriptor_type () override
  Interface function to return the descriptor type as string.
- int update_order (int order)
  Update the field expansion order.

Public Member Functions inherited from ParticleDescriptor

- ParticleDescriptor (GeometryConfiguration *gconf, ScattererConfiguration *sconf)
  ParticleDescriptor instance constructor.
- ParticleDescriptor (const ParticleDescriptor &rhs)
  ParticleDescriptor copy constructor.
- ~**ParticleDescriptor** ()
  ParticleDescriptor instance destroyer.

Static Public Member Functions

- static long get_size (GeometryConfiguration *gconf, ScattererConfiguration *sconf)
  Compute the memory requirements of an instance.

Static Public Member Functions inherited from ParticleDescriptor

- static long get_size (GeometryConfiguration *gconf, ScattererConfiguration *sconf)

  Compute the memory requirements of an instance.

Additional Inherited Members

Public Attributes inherited from ParticleDescriptor

- const short & **class_type** = _class_type

  Sub-class identification code.
- const int & **nsph** = _nsph

  Read-only view of number of spheres composing the model.
- const int & **li** = _li

  Read-only view of maximum internal field expansion order.
- const int & **max_layers** = _max_layers

  Read-only view of the maximum number of layers in types.
- const int & **num_configurations** = _num_configurations

  Read-only view of number of different sphere types.
- const int & **num_layers** = _num_layers

  Read-only view of total number of layers from all sphere types.
- const int & **nhspo** = _nhspo

  Read-only view of NHSPO.
- const int & **npnt** = _npnt

  Read-only view on number of points for numerical integration in layered spheres.
- const int & **npntts** = _npntts

  Read-only view on number of points for numerical integration in transition layer.
- dcomplex ** **rmi**

  Matrix of Mie scattering b-coefficients for different orders and spheres.
- dcomplex ** **rei**

  Matrix of Matrix of Mie scattering a-coefficients for different orders and spheres.
- dcomplex ** **w**

  Matrix of multipole amplitudes for incident and scattered fields.
- dcomplex * **vint**

  Vector of intensity components.
- double * **rxx**

  Vector of sphere Cartesian X coordinates.
- double * **ryy**

  Vector of sphere Cartesian Y coordinates.
- double * **rzz**

  Vector of sphere Cartesian Z coordinates.
- double * **ros**

  Vector of sphere radii.
- double ** **rc**

  Matrix of sphere fractional transition radii.
- int * **iog**

  Vector of sphere type identifiers.
- int * **nshl**

  Vector of number of layers in sphere type.
- dcomplex * **ris**

ANNOTATION: Square inverted coefficients.

- dcomplex * **dlri**

  ANNOTATION: Refractive index variation.

- dcomplex * **dc0**

  Vector of dielectric constants.

- dcomplex * **vkt**

  Vector of complex refractive indices.

- double * **vsz**

  *Vector of sizes in units of 2*PI/LAMBDA.*

- double **gcs**

  Total geometric cross-section.

- dcomplex *** **sas**

  Tensor of sphere scattering amplitudes.

- dcomplex ** **vints**

  Array of vectors of intensity components for single spheres.

- dcomplex * **fsas**

  Vector of sphere forward scattering amplitudes.

- double * **sscs**

  Vector of scattering cross-sections for spheres.

- double * **sexs**

  Vector of extinction cross-sections for spheres.

- double * **sabs**

  Vector of absorption cross-sections for spheres.

- double * **sqscs**

  Vector of scattering efficiencies for spheres.

- double * **sqexs**

  Vector of extinction efficiencies for spheres.

- double * **sqabs**

  Vector of absorption efficiencies for spheres.

- double * **gcsv**

  Vector of geometric cross-sections for spheres.

- dcomplex * **vintt**

  Vector of field intensity components.

- dcomplex **tfsas**

  Total forward scattering amplitude of the spheres.

- dcomplex ** **tsas**

  Total scattering amplitude of the spheres.

- double **scs**

  Sphere scattering cross-section.

- double **ecs**

  Sphere extinction cross-section.

- double **acs**

  Sphere absorption cross-section.

- dcomplex ** **gis**

  ANNOTATION: Geometric tensor.

- dcomplex ** **gls**

  ANNOTATION: Geometric tensor.

- dcomplex ** **sam**

  Mean scattering amplitude components.

- const int & **le** = _le

  Read-only view of maximum external field expansion order.

- const int & **lm** = _lm

  Read-only view of maximum field expansion order.
- const int & **nlim** = _nlim

  Read-only view of NLIM.
- const int & **nlem** = _nlem

  Read-only view of NLEM.
- const int & **nlemt** = _nlemt

  Read-only view of NLEMT.
- const int & **ncou** = _ncou

  Read-only view of NCOU.
- const int & **litpo** = _litpo

  Read-only view of LITPO.
- const int & **litpos** = _litpos

  Read-only view of LITPOS.
- const int & **lmpo** = _lmpo

  Read-only view of LMPO.
- const int & **lmtpo** = _lmtpo

  Read-only view of LMTPO.
- const int & **lmtpos** = _lmtpos

  Read-only view of LMTPOS.
- const int & **nv3j** = _nv3j

  Read-only view of NV3J.
- const int & **ndi** = _ndi

  Read-only view of NDI.
- const int & **ndit** = _ndit

  Read-only view of NDIT.
- const int & **ndm** = _ndm

  Read-only view of NDM.
- dcomplex * **vh**

  ANNOTATION: Hankel vector.
- dcomplex * **vj0**

  ANNOTATION: J0 vector.
- dcomplex * **vyhj**

  ANNOTATION: Translation vector.
- dcomplex * **vyj0**

  ANNOTATION: J0 translation vector.
- dcomplex **vj**

  ANNOTATION: J vector.
- dcomplex ** **am0m**

  Transition matrix.
- dcomplex ** **fsac**

  Cluster forward scattering amplitude.
- dcomplex ** **sac**

  Cluster scattering amplitude.
- dcomplex ** **fsacm**

  Mean cluster forward scattering amplitude.
- dcomplex * **vintm**

  Mean intensity components vector.
- dcomplex * **scscp**

  Cluster polarized scattering cross-sections.
- dcomplex * **ecscp**

Cluster polarized extinction cross-sections.

- dcomplex * **scscpm**

  Mean cluster polarized scattering cross-sections.

- dcomplex * **ecscpm**

  Mean cluster polarized extinction cross-sections.

- double * **v3j0**

  ANNOTATION: J0 vector.

- double * **scsc**

  Cluster scattering cross-sections.

- double * **ecsc**

  Cluster extinction cross-sections.

- double * **scscm**

  Mean cluster scattering cross-sections.

- double * **ecscm**

  Mean cluster extinction cross-sections.

- int ** **ind3j**

  J-vector components index matrix.

- double * **rac3j**

  ANNOTATION: J-vector boundary conditions.

- dcomplex * **rm0**

  ANNOTATION: M coefficients.

- dcomplex * **re0**

  ANNOTATION: E coefficients.

- dcomplex * **rmw**

  ANNOTATION: M amplitude coefficients.

- dcomplex * **rew**

  ANNOTATION: E amplitude coefficients.

- dcomplex * **tm**

  ANNOTATION: Transition M coefficients.

- dcomplex * **te**

  ANNOTATION: Transition E coefficients.

- dcomplex * **tm0**

  ANNOTATION: Initial transition M coefficients.

- dcomplex * **te0**

  ANNOTATION: Initial transition E coefficients.

- dcomplex ** **at**

  ANNOTATION: Included particle T-matrix.

Static Public Attributes inherited from ParticleDescriptor

- static const short **BASE_TYPE** = 0

  Base sub-class identification code.

- static const short **SPHERE_TYPE** = 1

  Sphere sub-class identification code.

- static const short **CLUSTER_TYPE** = 2

  Cluster sub-class identification code.

- static const short **INCLUSION_TYPE** = 3

  Inclusion sub-class identification code.

Protected Attributes inherited from ParticleDescriptor

- short **_class_type**

  Sub-class identification code.
- int **_nsph**

  Number of spheres composing the model.
- int **_li**

  Maximum internal field expansion order.
- int **_max_layers**

  Maximum number of layers in known sphere types.
- int **_num_configurations**

  Number of different sphere types.
- int **_num_layers**

  Total number of layers from all sphere types.
- int **_nhspo**

  NHSPO = 2 ∗ MAX(NPNT,NPNTTS) - 1.
- int **_npnt**

  Number of points for numerical integration in layered spheres.
- int **_npntts**

  Number of points for numerical integration in transition layer.
- dcomplex ∗ **vec_rmi**

  Contiguous space for RMI.
- dcomplex ∗ **vec_rei**

  Contiguous space for REI.
- dcomplex ∗ **vec_w**

  Contiguous space for W.
- double ∗ **vec_rc**

  Contiguous space for RC.
- dcomplex ∗ **vec_sas**

  Contiguous space for SAS.
- dcomplex ∗ **vec_vints**

  Contiguous space for VINTS.
- dcomplex ∗ **vec_tsas**

  Contiguous space for TSAS.
- dcomplex ∗ **vec_gis**

  Contiguous space for GIS.
- dcomplex ∗ **vec_gls**

  Contiguous space for GLS.
- dcomplex ∗ **vec_sam**

  Contiguous space for SAM.
- int **_le**

  Maximum external field expansion order.
- int **_lm**

  Maximum field expansion order.
- int **_nlim**

  NLIM = LI ∗ (LI + 2)
- int **_nlem**

  NLEM = LE ∗ (LE + 2)
- int **_nlemt**

  NLEMT = 2 ∗ NLEM.
- int **_ncou**

NCOU = NSPH ∗ NSPH - 1.

- int **_litpo**

  LITPO = 2 ∗ LI + 1.
- int **_litpos**

  LITPOS = LITPO ∗ LITPO.
- int **_lmpo**

  LMPO = LM + 1.
- int **_lmtpo**

  LMTPO = LI + LE + 1.
- int **_lmtpos**

  LMTPOS = LMTPO ∗ LMTPO.
- int **_nv3j**

  NV3J = (LM ∗ (LM + 1) ∗ (2 ∗ LM + 7)) / 6.
- int **_ndi**

  NDI = NSPH ∗ NLIM.
- int **_ndit**

  NDIT = 2 ∗ NSPH ∗ NLIM.
- dcomplex ∗ **vec_am0m**

  Contiguous space for AM0M.
- dcomplex ∗ **vec_fsac**

  Contiguous space for FSAC.
- dcomplex ∗ **vec_sac**

  Contiguous space for SAC.
- dcomplex ∗ **vec_fsacm**

  Contiguous space for FSACM.
- int ∗ **vec_ind3j**

  Contiguous space for IND3J.
- int **_ndm**

  NDM = NDIT + NLEMT.
- dcomplex ∗ **vec_at**

  Contiguous space for AT.

8.21.1 Detailed Description

This class is used to solve the problem of a single spherical particle. It replaces the old C1 class implementation of the corresponding FORTRAN common block.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 ParticleDescriptorSphere() [1/2]

```
ParticleDescriptorSphere::ParticleDescriptorSphere (
            GeometryConfiguration * gconf,
            ScattererConfiguration * sconf )
```

Parameters

gconf	`const GeometryConfiguration *` Pointer to GeometryConfiguration instance.
sconf	`const ScattererConfiguration *` Pointer to ScattererConfiguration instance.

8.21.2.2 ParticleDescriptorSphere() [2/2]

```
ParticleDescriptorSphere::ParticleDescriptorSphere (
            const ParticleDescriptorSphere & rhs )
```

Parameters

rhs	const ParticleDescriptorSphere & Reference to ParticleDescriptorSphere object to be copied.

8.21.3 Member Function Documentation

8.21.3.1 get_descriptor_type()

```
std::string ParticleDescriptorSphere::get_descriptor_type ( )  [inline], [override], [virtual]
```

Returns

descriptor_type: string The descriptor type name.

Reimplemented from ParticleDescriptor.

8.21.3.2 get_size()

```
long ParticleDescriptorSphere::get_size (
            GeometryConfiguration * gconf,
            ScattererConfiguration * sconf )  [static]
```

Parameters

gconf	GeometryConfiguration * Pointer to a GeometryConfiguration object.
sconf	ScattererConfiguration * Pointer to a ScattererConfiguration object.

Returns

result: long Estimated size in bytes.

8.21.3.3 update_order()

```
int ParticleDescriptorSphere::update_order (
            int order )
```

Parameters

order	int The new field expansion order to be set.

Returns

result: `int` An exit code (0 if successful).

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

- np_tmcode/src/include/Commons.h
- np_tmcode/src/libnptm/Commons.cpp

8.22 pydynrange.PlotData Class Reference

Class to represent the dynamic range memory structure.

Public Member Functions

- **__init__** (self)
  PlotData instance constructor.
- **log_dynamic_range** (self)
  Print a text log of the dynamic range.
- plot_dynamic_range (self, config)
  Make histograms of dynamic range with matplotlib.

Public Attributes

- **max_real**
  Maximum real part value.
- **max_imag**
  Maximum imaginary part value.
- **max_absl**
  Maximum absolute value.
- **min_real**
  Minimum (most negative) real part value.
- **min_imag**
  Minimum (most negative) imaginary part value.
- **sml_absl**
  Smallest absolute value.
- **sml_real**
  Smallest absolute real part value.
- **sml_imag**
  Smallest absolute imaginary part value.
- **absl_hist**
  Histogram of absolute values.
- **real_hist**
  Histogram of real values.
- **imag_hist**
  Histogram of imaginary values.
- **max_real_mag**
  Maximum real order of magnitude.

- **min_real_mag**

  Minimum real order of magnitude.
- **max_imag_mag**

  Maximum imaginary order of magnitude.
- **min_imag_mag**

  Minimum imaginary order of magnitude.
- **max_absl_mag**

  Maximum absolute order of magnitude.
- **min_absl_mag**

  Minimum absolute order of magnitude.

8.22.1 Detailed Description

In order to implement a light-weight test that is able to choose whether to produce a text-only output or a text + diagnostic plot output, the `PlotData` class stores the memory structures to produce the text log and the plots. The data for the plots are filled with dummy place-holders if the required output is text only, while they are filled with actual data, otherwise.

Returns

exit_code: `int` 0 on successful completion.

8.22.2 Member Function Documentation

8.22.2.1 plot_dynamic_range()

```
pydynrange.PlotData.plot_dynamic_range (
            self,
            config )
```

Parameters

config	`dict` Dictionary of configuration options.

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

- np_tmcode/src/scripts/pydynrange.py

8.23 ScattererConfiguration Class Reference

A class to represent scatterer configuration objects.

```
#include <Configuration.h>
```

Public Member Functions

- ScattererConfiguration (int nsph, int configs, double *scale_vector, int nxi, const std::string &variable_name, int *iog_vector, double *ros_vector, int *nshl_vector, double **rcf_vector, int dielectric_func_type, dcomplex ***dc_matrix, bool has_external, double exdc, double wp, double xip)

  Build a scatterer configuration structure.
- ScattererConfiguration (const ScattererConfiguration &rhs)

  Build a scatterer configuration structure copying its contents from a preexisting one.
- ~**ScattererConfiguration** ()

  Destroy a scatterer configuration instance.
- dcomplex get_dielectric_constant (int i, int j, int k)

  Get the dielectric constant of a material for a specific wavelength.
- int get_iog (int index)

  Get the ID of a configuration from the index of the sphere.
- double get_max_radius ()

  Get the maximum radius of the sphere components.
- double get_min_radius ()

  Get the minimum radius of the sphere components.
- int get_nshl (int index)

  Get the number of layers for a given configuration.
- double get_particle_radius (GeometryConfiguration *gc)

  Get the radius of the smallest sphere containing the particle.
- double get_radius (int index)

  Get the radius of a sphere by its index.
- double get_rcf (int row, int column)

  Get the value of a scale by its index.
- double get_scale (int index)

  Get the value of a scale by its index.
- double get_type_radius (int index)

  Get the radius of a sphere type by its index.
- void print ()

  Print the contents of the configuration object to terminal.
- void write_binary (const std::string &file_name, const std::string &mode="LEGACY")

  Write the scatterer configuration data to binary output.
- void write_formatted (const std::string &file_name)

  Write the scatterer configuration data to formatted text output.
- bool operator== (const ScattererConfiguration &other)

  Test whether two instances of ScattererConfiguration are equal.

Static Public Member Functions

- static ScattererConfiguration * from_binary (const std::string &file_name, const std::string &mode="LEGACY")

  Build configuration from binary configuration input file.
- static ScattererConfiguration * from_dedfb (const std::string &file_name)

  Build scatterer configuration from legacy configuration input file.

Public Attributes

- const std::string & **reference_variable_name** = _reference_variable_name

  Read-only view on name of the reference variable type.
- const int & **number_of_spheres** = _number_of_spheres

  Read-only view on number of spherical components.
- const int & **configurations** = _configurations

  Read-only view on number of configurations.
- const int & **number_of_scales** = _number_of_scales

  Read-only view on number of scales to use in calculation.
- const int & **idfc** = _idfc

  Read-only view on type of dielectric functions.
- const double & **exdc** = _exdc

  Read-only view on external medium dielectric constant.
- const double & **wp** = _wp

  Read-only view on WP.
- const double & **xip** = _xip

  Read-only view on peak XI.
- const int & **max_layers** = _max_layers

  Read-only view on the maximum number of layers for the particle components.
- const bool & **use_external_sphere** = _use_external_sphere

  Read-only view on flag to control whether to add an external layer.
- double ** **_rcf**

  Matrix of fractional transition radii with size [CONFIGURATIONS x LAYERS].

Protected Member Functions

- void write_hdf5 (const std::string &file_name)

  Write the scatterer configuration data to HDF5 binary output.
- void write_legacy (const std::string &file_name)

  Write the scatterer configuration data to legacy binary output.

Static Protected Member Functions

- static ScattererConfiguration * from_hdf5 (const std::string &file_name)

  Build configuration from a HDF5 binary input file.
- static ScattererConfiguration * from_legacy (const std::string &file_name)

  Build configuration from legacy binary input file.

Protected Attributes

- dcomplex *** **_dc0_matrix**

  Matrix of dielectric parameters with size [NON_TRANS_LAYERS x N_SPHERES x N_SCALES].
- double * **_radii_of_spheres**

  Vector of sphere radii expressed in m, with size [N_SPHERES].
- int * **_iog_vec**

  Vector of sphere ID numbers, with size [N_SPHERES].
- int * **_nshl_vec**

  Vector of layer numbers for every sphere, with size [CONFIGURATIONS].

- double * **_scale_vec**

  Vector of scale parameters, with size [N_SCALES].
- std::string **_reference_variable_name**

  Name of the reference variable type (one of XIV, WNS, WLS, PUS, EVS).
- int **_number_of_spheres**

  Number of spherical components.
- int **_configurations**

  Number of configurations.
- int **_number_of_scales**

  Number of scales to use in calculation.
- int **_idfc**

  Type of dielectric functions (<0 at XIP, =0 as function of XI, >0 constants).
- double **_exdc**

  External medium dielectric constant.
- double **_wp**

  WP. Peak pulsation.
- double **_xip**

  Peak scale.
- int **_max_layers**

  Maximum number of layers for the particle components.
- bool **_use_external_sphere**

  Flag to control whether to add an external layer.

8.23.1 Detailed Description

ScattererConfiguration is a class designed to store the necessary configuration data to describe the scatterer properties.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 ScattererConfiguration() [1/2]

```
ScattererConfiguration::ScattererConfiguration (
            int nsph,
            int configs,
            double * scale_vector,
            int nxi,
            const std::string & variable_name,
            int * iog_vector,
            double * ros_vector,
            int * nshl_vector,
            double ** rcf_vector,
            int dielectric_func_type,
            dcomplex *** dc_matrix,
            bool has_external,
            double exdc,
            double wp,
            double xip )
```

Prepare a default configuration structure by allocating the necessary memory structures.

Parameters

nsph	`int` The number of spheres in the simulation.
configs	`int` Number of spherical monometer configuration types.
scale_vector	`double*` The radiation-particle scale vector.
nxi	`int` The number of radiation-particle scalings.
variable_name	`string` The name of the radiation-particle scaling type.
iog_vector	`int*` Array of sphere identification numbers.
ros_vector	`double*` Sphere radius array.
nshl_vector	`int*` Array of layer numbers.
rcf_vector	`double**` Array of fractional break radii.
dielectric_func_type	`int` Type of dielectric function definition (=0 for constant, >0 as function of scale parameter, <0 for functions at XIP value and XI is scale factor for dimensions).
dc_matrix	`complex double ***` Matrix of reference dielectric constants.
has_external	`bool` Flag to set whether to add an external spherical layer.
exdc	`double` External medium dielectric constant.
wp	`double` wp
xip	`double` xip

8.23.2.2 ScattererConfiguration() [2/2]

```
ScattererConfiguration::ScattererConfiguration (
            const ScattererConfiguration & rhs )
```

Prepare a default configuration structure by allocating the necessary memory structures.

Parameters

rhs	`ScattererConfiguration&` Reference to the ScattereConfiguration object to be copied.

8.23.3 Member Function Documentation

8.23.3.1 from_binary()

```
ScattererConfiguration * ScattererConfiguration::from_binary (
            const std::string & file_name,
            const std::string & mode = "LEGACY" )  [static]
```

The configuration step can save configuration data as a binary file. The original FORTRAN code used this possibility to manage communication between the configuring code and the calculation program. This possibility is maintained, in case the configuration step needs to be separated from the calculation execution. In this case, `from_binary()` is the class method that restores a ScattererConfiguration object from a previously saved binary file.

Parameters

file_name	`string` Name of the binary configuration data file.
mode	`string` Binary encoding. Can be one of "LEGACY", ... Optional (default is "LEGACY").

Returns

config: `ScattererConfiguration*` Pointer to object containing the scatterer configuration data.

8.23.3.2 from_dedfb()

```
ScattererConfiguration * ScattererConfiguration::from_dedfb (
            const std::string & file_name )  [static]
```

To allow for consistency tests and backward compatibility, ScattererConfiguration objects can be built from legacy configuration files. This function replicates the approach implemented by the FORTRAN EDFB code, but using a C++ oriented work-flow.

Parameters

file_name	`string` Name of the legacy configuration data file.

Returns

config: `ScattererConfiguration*` Pointer to object containing the scatterer configuration data.

8.23.3.3 from_hdf5()

```
ScattererConfiguration * ScattererConfiguration::from_hdf5 (
            const std::string & file_name )  [static], [protected]
```

This is the function called by the public method `from_binary()` in case of HDF5 mode selection. This function creates a configuration structure from a binary file written according to the HDF5 format standard.

Parameters

file_name	`string` Name of the binary configuration data file.

Returns

config: `ScattererConfiguration*` Pointer to object containing the scatterer configuration data.

8.23.3.4 from_legacy()

```
ScattererConfiguration * ScattererConfiguration::from_legacy (
            const std::string & file_name )  [static], [protected]
```

This is the function called by the public method `from_binary()` in case of legacy mode selection. This function creates a configuration structure from a binary file written according to the proprietary mode used by the original FORTRAN code.

Parameters

file_name	`string` Name of the binary configuration data file.

Returns

config: `ScattererConfiguration*` Pointer to object containing the scatterer configuration data.

8.23.3.5 get_dielectric_constant()

```
dcomplex ScattererConfiguration::get_dielectric_constant (
            int i,
            int j,
            int k )  [inline]
```

Dielectric constants are stored in a 3D complex matrix, whose dimensions map to [NUMBER_OF_↩ CONFIGURATIONS x NUMBER_OF_SPHERES x NUMBER_OF_SCALES]. This function extracts such values from the matrix through their indices.

Parameters

i	`int` Index of the configuration.
j	`int` Index of the sphere.
k	`int` Index of the current scale.

Returns

radius: `dcomplex` The requested dielectric constant.

8.23.3.6 get_iog()

```
int ScattererConfiguration::get_iog (
            int index )  [inline]
```

This is a specialized function to get a configuration ID through the index of the sphere it applies to.

Parameters

index	`int` Index of the sphere.

Returns

ID: `int` ID of the configuration to be applied.

8.23.3.7 get_max_radius()

```
double ScattererConfiguration::get_max_radius ( )
```

Returns

radius: `double` The radius of the largest sphere.

8.23.3.8 get_min_radius()

```
double ScattererConfiguration::get_min_radius ( )
```

Returns

radius: `double` The radius of the smallest sphere.

8.23.3.9 get_nshl()

```
int ScattererConfiguration::get_nshl (
            int index )  [inline]
```

This is a specialized function to get the number of layers in a specific configuration.

Parameters

index	`int` Index of the configuration.

Returns

nl: `int` The number of layers for the given configuration.

8.23.3.10 get_particle_radius()

```
double ScattererConfiguration::get_particle_radius (
            GeometryConfiguration * gc )
```

Parameters

gc	`GeometryConfiguration *` Pointer to a `GeometryConfiguration` instance.

Returns

radius: `double` The radius of the sphere containing the particle.

8.23.3.11 get_radius()

```
double ScattererConfiguration::get_radius (
            int index )
```

This is a specialized function to get the radius of a sphere through its index.

Parameters

index	`int` Index of the ID to be retrieved.

Returns

radius: `double` The requested sphere radius.

8.23.3.12 get_rcf()

```
double ScattererConfiguration::get_rcf (
            int row,
            int column )  [inline]
```

This is a specialized function to access a scale (generally a wavelength), through its index.

Parameters

row	`int` Row index of the element to be retrieved.
column	`int` Column index of the element to be retrieved.

Returns

scale: `double` The desired scale.

8.23.3.13 get_scale()

```
double ScattererConfiguration::get_scale (
            int index )  [inline]
```

This is a specialized function to access a scale (generally a wavelength), through its index.

Parameters

index	`int` Index of the scale to be retrieved.

Returns

scale: `double` The desired scale.

8.23.3.14 get_type_radius()

```
double ScattererConfiguration::get_type_radius (
            int index )  [inline]
```

This is a specialized function to get the radius of a sphere type through its index. Sphere types range from 0 to NUM_CONFIGURATIONS - 1.

Parameters

index	`int` Index of the ID to be retrieved.

Returns

radius: `double` The requested sphere radius.

8.23.3.15 operator==()

```
bool ScattererConfiguration::operator== (
            const ScattererConfiguration & other )
```

ScattererConfiguration objects can be obtained in a variety of manner. They can be constructed manually, through the class constructor, they can be read from formatted configuration files, or they can be loaded from binary files. The == operator tests for the equivalence of two configuration instances, returning `true` if they are equivalnet, `false` otherwise.

Parameters

other	`ScattererConfiguration &` Reference to the instance to be compared with.

Returns

result: `bool` True, if the two instances are equal, false otherwise.

8.23.3.16 print()

```
void ScattererConfiguration::print ( )
```

In case of quick debug testing, `ScattererConfiguration.print()` allows printing a formatted summary of the configuration data to terminal.

8.23.3.17 write_binary()

```
void ScattererConfiguration::write_binary (
            const std::string & file_name,
            const std::string & mode = "LEGACY" )
```

The execution work-flow may be split in a configuration step and one or more calculation steps. In case the calculation is not being run all-in-one, it can be useful to save the configuration data. `ScattererConfiguration.write_binary()` performs the operation of saving the configuration in binary format. This function can work in legacy mode, to write backward compatible configuration files, as well as by wrapping the data into common scientific formats.

Parameters

file_name	`string` Name of the file to be written.
mode	`string` Binary encoding. Can be one of ["LEGACY", "HDF5"] . Optional (default is "LEGACY").

8.23.3.18 write_formatted()

```
void ScattererConfiguration::write_formatted (
```

```
            const std::string & file_name )
```

Writing configuration to formatted text is an optional operation, which may turn out to be useful for consistency checks. As a matter of fact, formatted configuration output is not read back by the FORTRAN code work-flow and it can be safely omitted, unless there is a specific interest in assessing that the legacy code and the updated one are doing the same thing.

Parameters

file_name	`string` Name of the file to be written.

8.23.3.19 write_hdf5()

```
void ScattererConfiguration::write_hdf5 (
            const std::string & file_name )  [protected]
```

This function is invoked by the public method `write_binary()` with the "HDF5" format mode. It undertakes the task of writing the configuration information to a binary file using the standard HDF5 format.

Parameters

file_name	`string` Name of the binary configuration data file.

8.23.3.20 write_legacy()

```
void ScattererConfiguration::write_legacy (
            const std::string & file_name )  [protected]
```

This function is invoked by the public method `write_binary()` with the "LEGACY" format mode. It undertakes the task of writing the configuration information to a binary file using a proprietary format, as it was done originally in the FORTRAN code.

Parameters

file_name	`string` Name of the binary configuration data file.

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

- np_tmcode/src/include/Configuration.h
- np_tmcode/src/libnptm/Configuration.cpp

8.24 ScatteringAngles Class Reference

A data structure representing the angles to be evaluated in the problem.

```
#include <Commons.h>
```

Public Member Functions

- ScatteringAngles (GeometryConfiguration *gconf)

  ScatteringAngles instance constructor.
- ScatteringAngles (const ScatteringAngles &rhs)

  ScatteringAngles copy constructor.

Public Attributes

- const int & **nth** = _nth

  Read only view of _nth.
- const int & **nths** = _nths

  Read only view of _nths.
- const int & **nph** = _nph

  Read only view of _nph.
- const int & **nphs** = _nphs

  Read only view of _nphs.
- const int & **nk** = _nk

  Read only view of _nk.
- const int & **nks** = _nks

  Read only view of _nks.
- const int & **nkks** = _nkks

  Read only view of _nkks.
- const double & **th** = _th

  Read only view of _th.
- const double & **thstp** = _thstp

  Read only view of _thstp.
- const double & **thlst** = _thlst

  Read only view of _thlst.
- const double & **ths** = _ths

  Read only view of _ths.
- const double & **thsstp** = _thsstp

  Read only view of _thsstp.
- const double & **thslst** = _thslst

  Read only view of _thslst.
- const double & **ph** = _ph

  Read only view of _ph.
- const double & **phstp** = _phstp

  Read only view of _phstp.
- const double & **phlst** = _phlst

  Read only view of _phlst.
- const double & **phs** = _phs

  Read only view of _phs.
- const double & **phsstp** = _phsstp

  Read only view of _phsstp.
- const double & **phslst** = _phslst

  Read only view of _phslst.
- const double & **thsca** = _thsca

  Read only view of _thsca.

Protected Attributes

- int **_nth**
  Number of incident field azimuth angles.
- int **_nths**
  Number of scattered field azimuth angles.
- int **_nph**
  Number of incident field elevation angles.
- int **_nphs**
  Number of scattered field elevation angles.
- int **_nk**
  Number of incident field propagation angles.
- int **_nks**
  Number of scattered field propagation angles.
- int **_nkks**
  Total number of field propagation angles.
- double **_th**
  First incident field azimuth angle.
- double **_thstp**
  Incident field azimuth angle increment.
- double **_thlst**
  Last incident field azimuth angle.
- double **_ths**
  First scattered field azimuth angle.
- double **_thsstp**
  Scattered field azimuth angle increment.
- double **_thslst**
  Last scattered field azimuth angle.
- double **_ph**
  First incident field elevation angle.
- double **_phstp**
  Incident field elevation angle increment.
- double **_phlst**
  Last incident field elevation angle.
- double **_phs**
  First scattered field elevation angle.
- double **_phsstp**
  Scattered field elevation angle increment.
- double **_phslst**
  Last scattered field elevation angle.
- double **_thsca**
  Azimuth scattering deflection.

8.24.1 Constructor & Destructor Documentation

8.24.1.1 ScatteringAngles() [1/2]

```
ScatteringAngles::ScatteringAngles (
            GeometryConfiguration * gconf )
```

Parameters

gconf	`GeometryConfiguration*` Pointer to a GeometryConfiguration object.

8.24.1.2 ScatteringAngles() [2/2]

```
ScatteringAngles::ScatteringAngles (
            const ScatteringAngles & rhs )
```

Parameters

rhs	`ScatteringAngles&` Reference to the ScatteringAngles object to be copied.

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

- np_tmcode/src/include/Commons.h
- np_tmcode/src/libnptm/Commons.cpp

8.25 SphereIterationData Class Reference

A data structure representing the information used for a single scale of the SPHERE case.

`#include <IterationData.h>`

Public Member Functions

- SphereIterationData (GeometryConfiguration *gconf, ScattererConfiguration *sconf, const mixMPI *mpidata, const int device_count)

  SphereIterationData default instance constructor.
- SphereIterationData (const SphereIterationData &rhs)

  SphereIterationData copy constructor.
- ~**SphereIterationData** ()

  SphereIterationData instance destroyer.
- int update_order (int order)

  Update field expansion order.

Public Attributes

- double **cost**

  Cosine of incident radiation azimuth.
- double **sint**

  Sine of incident radiation azimuth.
- double **cosp**

  Cosine of incident radiation elevation.
- double **sinp**

  Sine of incident radiation elevation.

- double **costs**

  Cosine of scattered radiation azimuth.
- double **sints**

  Sine of scattered radiation azimuth.
- double **cosps**

  Cosine of scattered radiation elevation.
- double **sinps**

  Sine of scattered radiation elevation.
- double **vk**

  Vacuum magnitude of wave vector.
- double **wn**

  Wave number.
- double **xip**

  ANNOTATION: Normalization scale.
- int **number_of_scales**

  Number of scales (wavelengths) to be computed.
- int **xiblock**

  Size of the block of scales handled by the current process.
- int **firstxi**

  Index of the first scale handled by the current process.
- int **lastxi**

  Index of the last scale handled by the current process.
- dcomplex **arg**

  Argument of harmonic functions.
- dcomplex **s0**

  S0 = FSAS / (4 PI K^ 3).
- dcomplex **tfsas**

  Total forward scattering amplitude of the spheres.
- ParticleDescriptor * **c1**

  Pointer to a sphere particle descriptor.
- double * **argi**

  Imaginary part of `arg`.
- double * **args**

  `arg` *squared.*
- double **scan**

  Scattering angle.
- double **cfmp**

  Control parameter on incidence direction referred to meridional plane.
- double **sfmp**

  Control parameter on scattering direction referred to meridional plane.
- double **cfsp**

  Control parameter on incidence direction referred to scattering plane.
- double **sfsp**

  Control parameter on scattering direction referred to scattering plane.
- double * **gaps**

  Geometry asymmetry parameter for spheres.
- double * **duk**

  Variation of unitary wave vector.
- double * **u**

  Incidence direction unitary vector.
- double * **us**

Scattering direction unitary vector.

- double * **un**

  Normal direction unitary vector.

- double * **uns**

  Scattering normal direction unitary vector.

- double * **up**

  Polarization direction unitary vector.

- double * **ups**

  Scattered polarization direction unitary vector.

- double * **upmp**

  Polarization direction unitary vector referred to meridional plane.

- double * **upsmp**

  Scattered polarization direction unitary vector referred to meridional plane.

- double * **unmp**

  Normal direction unitary vector referred to meridional plane.

- double * **unsmp**

  Scattering normal direction unitary vector referred to meridional plane.

- double ** **cmul**

  Mueller matrix components.

- double ** **cmullr**

  Mueller matrix components referred to meridional plane.

- dcomplex ** **tqspe**

  Polarization-dependent extinction contribution to torque for each sphere.

- dcomplex ** **tqsps**

  Polarization-dependent scattering contribution to torque for each sphere.

- double ** **tqse**

  Extinction contribution to torque for each sphere.

- double ** **tqss**

  Scattering contribution to torque for each sphere.

- double **** **zpv**

  Scattering coefficients tensor.

- bool **is_first_scale**

  flag for first time initialisation

Protected Attributes

- int **_nsph**

  Number of spheres.

- int **_lm**

  Maximum field expansion order.

- double * **vec_cmul**

  Vector of Mueller matrix components.

- double * **vec_cmullr**

  Vector of Mueller matrix components referred to meridional plane.

- dcomplex * **vec_tqspe**

  Vectorized TQSPE.

- dcomplex * **vec_tqsps**

  Vectorized TQSPS.

- double * **vec_tqse**

  Vectorized TQSE.

- double * **vec_tqss**

  Vectorized TQSS.

- double * **vec_zpv**

  Vectorized ZPV.

8.25.1 Constructor & Destructor Documentation

8.25.1.1 SphereIterationData() [1/2]

```
SphereIterationData::SphereIterationData (
            GeometryConfiguration * gconf,
            ScattererConfiguration * sconf,
            const mixMPI * mpidata,
            const int device_count )
```

Parameters

gconf	`GeometryConfiguration *` Pointer to a `GeometryConfiguration` object.
sconf	`ScattererConfiguration *` Pointer to a `ScattererConfiguration` object.
mpidata	`mixMPI *` Pointer to a `mixMPI` object.
device_count	`const int` Number of offload devices available on the system.

8.25.1.2 SphereIterationData() [2/2]

```
SphereIterationData::SphereIterationData (
            const SphereIterationData & rhs )
```

Parameters

rhs	`const SphereIterationData &` Reference to the object to be copied.

8.25.2 Member Function Documentation

8.25.2.1 update_order()

```
int SphereIterationData::update_order (
            int order )
```

Parameters

order	`int` The new expansion order to be set.

Returns

result: `int` An exit code (0 if successful).

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

- np_tmcode/src/include/IterationData.h
- np_tmcode/src/sphere/sphere.cpp

8.26 SphereOutputInfo Class Reference

Class to collect output information for scattering from a single sphere.

`#include <outputs.h>`

Public Member Functions

- SphereOutputInfo (ScattererConfiguration *sc, GeometryConfiguration *gc, const mixMPI *mpidata, int first_xi=1, int xi_length=0)

  SphereOutputInfo default instance constructor.
- SphereOutputInfo (const std::string &hdf5_name)

  SphereOutputInfo constructor from HDF5 input.
- SphereOutputInfo (const int skip_flag)

  SphereOutputInfo constructor for the dummy NULL case
- ~**SphereOutputInfo** ()

  SphereOutputInfo instance destroyer.
- long compute_size ()

  Get the size of a SphereOutputInfo instance in bytes.
- int insert (const SphereOutputInfo &rhs)

  Insert in the current output data the data of another block.
- int write (const std::string &output, const std::string &format)

  Write the output to a file.

Static Public Member Functions

- static long compute_size (ScattererConfiguration *sc, GeometryConfiguration *gc, int first_xi=1, int xi_length=0)

  Estimate the size of the structure that would be built for given input.

Public Attributes

- const int & **skip_flag** = _skip_flag

  Read-only view on skip_flag.
- const int & **first_xi** = _first_xi

  Read-only view on the ID of the first scale.
- int **nsph**

  Number of spheres.
- int **lm**

  Maximum field expansion order.
- int **inpol**

  Incident polarization flag.
- int **npnt**

  Number of points for transition layer integration.
- int **npntts**

  Number of points for non-transition layer integration.
- int **isam**

  Flag for reference to meridional plane.
- int **idfc**

Flag for dielectric function definition.

- double **th**

  First incident radiation azimuth angle.
- double **thstp**

  Incident radiation azimuth angle step.
- double **thlst**

  Last incident radiation azimuth angle.
- double **ths**

  First scattered radiation azimuth angle.
- double **thsstp**

  Scattered radiation azimuth angle step.
- double **thslst**

  Last scattered radiation azimuth angle.
- double **ph**

  First incident radiation elevation angle.
- double **phstp**

  Incident radiation elevation angle step.
- double **phlst**

  Last incident radiation elevation angle.
- double **phs**

  First scattered radiation elevation angle.
- double **phsstp**

  Scattered radiation elevation angle step.
- double **phslst**

  Last scattered radiation elevation angle.
- int **ndirs**

  Number of directions to be explicitly solved.
- double **exri**

  Refractive index of external medium.
- int **nxi**

  Number of scales (wavelengths)
- int **xi_block_size**

  Number of scales handled by the current process.
- int **jwtm**

  Index of the wavelength for T-matrix output.
- int **configurations**

  Number of sphere types.
- int **lcalc**

  Highest expansion order achieved in calculations.
- dcomplex **arg**

  Harmonic functions argument.
- int * **vec_jxi**

  Vector of scale (wavelength) indices.
- short * **vec_ier**

  Vector of error severities (0 - success, 1 - DME).
- double * **vec_vk**

  Vector of vacuum wave numbers.
- double * **vec_xi**

  Vector of computed scales.
- double * **vec_sphere_sizes**

  Vector of sphere sizes (one for every configuration and scale).

- dcomplex * **vec_sphere_ref_indices**

  Vector of sphere refractive indices (one for every configuration and scale).
- double * **vec_scs**

  Vector of sphere scattering cross-sections.
- double * **vec_abs**

  Vector of sphere absorption cross-sections.
- double * **vec_exs**

  Vector of sphere extinction cross-sections.
- double * **vec_albeds**

  Vector of sphere albedos.
- double * **vec_scsrt**

  Vector of sphere scattering-to-geometric cross-sections.
- double * **vec_absrt**

  Vector of sphere absorption-to-geometric cross-sections.
- double * **vec_exsrt**

  Vector of sphere extinction-to-geometric cross-sections.
- dcomplex * **vec_fsas**

  Vector of sphere forward scattering amplitudes.
- double * **vec_qschu**

  Vector of sphere QSCHU.
- double * **vec_pschu**

  Vector of sphere PSCHU.
- double * **vec_s0mag**

  Vector of sphere S0MAG.
- double * **vec_cosav**

  Vector of sphere average asymmetry parameter.
- double * **vec_raprs**

  Vector of sphere average radiation pressure force (N).
- double * **vec_tqek1**

  Vector of sphere average extinction torque along incidence direction (parallel polarization).
- double * **vec_tqek2**

  Vector of sphere average extinction torque along incidence direction (perpendicular polarization).
- double * **vec_tqsk1**

  Vector of sphere average scattering torque along incidence direction (parallel polarization).
- double * **vec_tqsk2**

  Vector of sphere average scattering torque along incidence direction (perpendicular polarization).
- dcomplex * **vec_fsat**

  Vector of total forward scattering amplitudes.
- double * **vec_qschut**

  Vector of total QSCHU.
- double * **vec_pschut**

  Vector of total PSCHU.
- double * **vec_s0magt**

  Vector of total S0MAG.
- double * **vec_dir_tidg**

  Vector of incidence azimuth directions (one per incidence azimuth).
- double * **vec_dir_pidg**

  Vector of incidence elevation directions (one per incidence elevation).
- double * **vec_dir_tsdg**

  Vector of scattering azimuth directions (one per scattering azimuth).
- double * **vec_dir_psdg**

Vector of scattering elevation directions (one per scattering elevation).

- double * **vec_dir_scand**

  Vector of scattering angles (one per direction).
- double * **vec_dir_cfmp**

  Control parameter for incidence plane referred to meridional plane (one per direction).
- double * **vec_dir_sfmp**

  Control parameter for scattering plane referred to meridional plane (one per direction).
- double * **vec_dir_cfsp**

  Control parameter for incidence plane referred to scattering plane (one per direction).
- double * **vec_dir_sfsp**

  Control parameter for scattering plane referred to scattering plane (one per direction).
- double * **vec_dir_un**

  Components of the unitary vector perpendicular to incidence plane (three per direction).
- double * **vec_dir_uns**

  Components of the unitary vector perpendicular to scattering plane (three per direction).
- dcomplex * **vec_dir_sas11**

  Vector of sphere differential scattering amplitude with polarization parallel to parallel incidence field.
- dcomplex * **vec_dir_sas21**

  Vector of sphere differential scattering amplitude with polarization perpendicular to the parallel incidence field.
- dcomplex * **vec_dir_sas12**

  Vector of sphere differential scattering amplitude with polarization perpendicular to perpendicular incidence field.
- dcomplex * **vec_dir_sas22**

  Vector of sphere differential scattering amplitude with polarization parallel the perpendicular incidence field.
- double * **vec_dir_fx**

  Vector of differential radiation pressure force components along the X axis.
- double * **vec_dir_fy**

  Vector of differential radiation pressure force components along the Y axis.
- double * **vec_dir_fz**

  Vector of differential radiation pressure force components along the Z axis.
- double * **vec_dir_muls**

  Vector of sphere Mueller transormation matrices referred to meridional plane.
- double * **vec_dir_mulslr**

  Vector of sphere Mueller transormation matrices referred to scattering plane.

Protected Member Functions

- int write_hdf5 (const std::string &file_name)

  Write the output to a HDF5 file.
- int write_legacy (const std::string &output)

  Write the output to a legacy text file.

Protected Attributes

- int **_skip_flag**

  Flag for skipping mpisend() and mpireceive()
- int **_num_theta**

  Number of incident azimuth calculations.
- int **_num_thetas**

  Number of scattered azimuth calculations.
- int **_num_phi**

Number of incident elevation calculations.

- int **_num_phis**

  Number of scattered elevation calculations.

- int **_first_xi**

  ID of the first computed wavelength.

8.26.1 Detailed Description

The results of the calculation can be saved in different formats. It is therefore convenient to have a proper memory structure that allows for storing the results and flushing them in any of the permitted formats with just one operation. The purpose of the `SphereOutputInfo` class is to provide a wrapper for the output of the spherical particle scattering solver.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 SphereOutputInfo() [1/3]

```
SphereOutputInfo::SphereOutputInfo (
            ScattererConfiguration * sc,
            GeometryConfiguration * gc,
            const mixMPI * mpidata,
            int first_xi = 1,
            int xi_length = 0 )
```

Parameters

sc	`ScattererConfiguration` * Pointer to a `ScattererConfiguration` instance.
gc	`GeometryConfiguration` * Pointer to a `GeometryConfiguration` instance.
mpidata	`const mixMPI*` Pointer to a mixMPI instance.
first_xi	`int` Index of the first scale in output (optional, default is 1).
xi_length	`int` Number of scales tobe included in output (optional, default is 0, meaning all).

8.26.2.2 SphereOutputInfo() [2/3]

```
SphereOutputInfo::SphereOutputInfo (
            const std::string & hdf5_name )
```

Parameters

hdf5_name	`const string &` Path to the HDF5 file to be read.

8.26.2.3 SphereOutputInfo() [3/3]

```
SphereOutputInfo::SphereOutputInfo (
            const int skip_flag )
```

Parameters

skip_flag	`const int` must be passed as the `1` constant.

8.26.3 Member Function Documentation

8.26.3.1 compute_size() [1/2]

```
long SphereOutputInfo::compute_size ( )
```

Returns

size: `long` Estimated instance size in bytes.

8.26.3.2 compute_size() [2/2]

```
long SphereOutputInfo::compute_size (
            ScattererConfiguration * sc,
            GeometryConfiguration * gc,
            int first_xi = 1,
            int xi_length = 0 )  [static]
```

Parameters

sc	`ScattererConfiguration *` Pointer to a `ScattererConfiguration` instance.
gc	`GeometryConfiguration *` Pointer to a `GeometryConfiguration` instance.
first_xi	`int` Index of the first scale in output (optional, default is 1).
xi_length	`int` Number of scales tobe included in output (optional, default is all).

Returns

size: `long` Estimated instance size in bytes.

8.26.3.3 insert()

```
int SphereOutputInfo::insert (
            const SphereOutputInfo & rhs )
```

Parameters

rhs	`const SphereOutputInfo &` Reference to the source data block.

Returns

result: `int` Exit code (0 if successful).

8.26.3.4 write()

```
int SphereOutputInfo::write (
            const std::string & output,
            const std::string & format )
```

Parameters

output	`const string &` Path to the output to be written.
format	`const string &` Output format (one of LEGACY or HDF5).

Returns

result: `int` Exit code (0 if successful).

8.26.3.5 write_hdf5()

```
int SphereOutputInfo::write_hdf5 (
            const std::string & file_name )  [protected]
```

Parameters

file_name	`const string &` Path to the output to be written.

Returns

result: `int` Exit code (0 if successful).

8.26.3.6 write_legacy()

```
int SphereOutputInfo::write_legacy (
            const std::string & output )  [protected]
```

This function takes care of writing the output using the legacy formatted ASCII structure. If the output file does not exist, it is created. If it exists, the new content is overwritten.

Parameters

output	`const string &` Path to the output to be written.

Returns

result: `int` Exit code (0 if successful).

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

- np_tmcode/src/include/outputs.h
- np_tmcode/src/libnptm/outputs.cpp

8.27 Swap1 Class Reference

Class to represent the first group of trapping swap data.

`#include <tfrfme.h>`

Public Member Functions

- Swap1 (int lm, int nkv)
  Swap1 instance constructor.
- ~**Swap1** ()
  Swap1 instance destroyer.
- void append (dcomplex value)
  Append an element at the end of the vector.
- void **reset** ()
  Bring the pointer to the next element at the start of vector.
- void write_binary (const std::string &file_name, const std::string &mode="LEGACY")
  Write a Swap1 instance to binary file.
- bool operator== (Swap1 &other)
  Test whether two instances of Swap1 are equal.

Static Public Member Functions

- static Swap1 * from_binary (const std::string &file_name, const std::string &mode="LEGACY")
  Load a Swap1 instance from binary file.
- static long get_size (int lm, int nkv)
  Calculate the necessary amount of memory to create a new instance.

Public Attributes

- const dcomplex * **wk**
  Read only view on WK.

Protected Member Functions

- void write_hdf5 (const std::string &file_name)
  Save a Swap1 instance to a HDF5 binary file.
- void write_legacy (const std::string &file_name)
  Save a Swap1 instance to a legacy binary file.

Static Protected Member Functions

- static Swap1 * from_hdf5 (const std::string &file_name)
  Load a Swap1 instance from a HDF5 binary file.
- static Swap1 * from_legacy (const std::string &file_name)
  Load a Swap1 instance from a legacy binary file.

Protected Attributes

- int **_last_index**

  Index of the last element to be filled.
- int **_nkv**

  Number of beam description wave-numbers.
- int **_nlmmt**

  NLMMT = 2 ∗ LM ∗ (LM + 2)
- dcomplex ∗ **_wk**

  ANNOTATION: incident amplitudes.

8.27.1 Constructor & Destructor Documentation

8.27.1.1 Swap1()

```
Swap1::Swap1 (
            int lm,
            int nkv )
```

Parameters

lm	`int` Maximum field expansion order.
nkv	`int` Number of beam description wave numbers.

8.27.2 Member Function Documentation

8.27.2.1 append()

```
void Swap1::append (
            dcomplex value )  [inline]
```

Parameters

value	`complex double` The value to be added to the vector.

8.27.2.2 from_binary()

```
Swap1 * Swap1::from_binary (
            const std::string & file_name,
            const std::string & mode = "LEGACY" )  [static]
```

Parameters

file_name	`string` Name of the file.
mode	`string` Format of the file (can be either "HDF5" or "LGEACY". Default is "LEGACY").

Returns

instance: `Swap1 *` Pointer to a newly created Swap1 instance.

8.27.2.3 from_hdf5()

```
Swap1 * Swap1::from_hdf5 (
            const std::string & file_name )  [static], [protected]
```

Parameters

file_name	`string` Name of the file to be loaded.

Returns

instance: `Swap1 *` Pointer to a new Swap1 instance.

8.27.2.4 from_legacy()

```
Swap1 * Swap1::from_legacy (
            const std::string & file_name )  [static], [protected]
```

Parameters

file_name	`string` Name of the file to be loaded.

Returns

instance: `Swap1 *` Pointer to a new Swap1 instance.

8.27.2.5 get_size()

```
long Swap1::get_size (
            int lm,
            int nkv )  [static]
```

Parameters

lm	`int` Maximum field expansion order.
nkv	`int` ANNOTATION: Number of wave vectors.

Returns

size: `long` The necessary memory size in bytes.

8.27.2.6 operator==()

```
bool Swap1::operator== (
            Swap1 & other )
```

Parameters

other	`Swap1 &` Reference to the instance to be compared with.

Returns

result: `bool` True, if the two instances are equal, false otherwise.

8.27.2.7 write_binary()

```
void Swap1::write_binary (
            const std::string & file_name,
            const std::string & mode = "LEGACY" )
```

Parameters

file_name	`string` Name of the file.
mode	`string` Format of the file (can be either "HDF5" or "LGEACY". Default is "LEGACY").

8.27.2.8 write_hdf5()

```
void Swap1::write_hdf5 (
            const std::string & file_name )  [protected]
```

Parameters

file_name	`string` Name of the file to be written.

8.27.2.9 write_legacy()

```
void Swap1::write_legacy (
            const std::string & file_name )  [protected]
```

Parameters

file_name	`string` Name of the file to be written.

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

- np_tmcode/src/include/tfrfme.h
- np_tmcode/src/libnptm/tfrfme.cpp

8.28 Swap2 Class Reference

Class to represent the second group of trapping swap data.

`#include <tfrfme.h>`

Public Member Functions

- Swap2 (int nkv)
  Swap2 instance constructor.
- ~**Swap2** ()
  Swap2 instance destroyer.
- double * get_vector ()
  Get the pointer to the VKV vector.
- void push_matrix (double value)
  Append an element at the end of the matrix.
- void push_vector (double value)
  Append an element at the end of the vector.
- void **reset_matrix** ()
  Bring the matrix pointer to the start of the array.
- void **reset_vector** ()
  Bring the vector pointer to the start of the array.
- void set_param (const std::string &param_name, double value)
  Set a parameter by its name and value.
- void write_binary (const std::string &file_name, const std::string &mode="LEGACY")
  Write a Swap2 instance to binary file.
- bool operator== (Swap2 &other)
  Test whether two instances of Swap2 are equal.

Static Public Member Functions

- static Swap2 * from_binary (const std::string &file_name, const std::string &mode="LEGACY")
  Load a Swap2 instance from binary file.
- static long get_size (int nkv)
  Calculate the necessary amount of memory to create a new instance.

Public Attributes

- const int & **last_vector** = _last_vector
  Read-only view onthe index of the last vector element to be filled.
- const int & **last_matrix** = _last_matrix
  Read-only view on the index of the last matrix element to be filled.
- const int & **nkv** = _nkv
  Read-only view on the number of beam description wave numbers.
- double * **vkv**
  ANNOTATION: wave number vector.
- double * **vec_vkzm**
  ANNOTATION: Vectorized VKZ matrix.
- const double & **apfafa** = _apfafa

ANNOTATION: Normalized aperture.

- const double & **pmf** = _pmf

  ANNOTATION: Aperture correction.

- const double & **spd** = _spd

  ANNOTATION: Cover slip thickness.

- const double & **rir** = _rir

  ANNOTATION: External over oil refractive index.

- const double & **ftcn** = _ftcn

  ANNOTATION: Refraction factor.

- const double & **fshmx** = _fshmx

  ANNOTATION: Cover slip correction factor.

- const double & **vxyzmx** = _vxyzmx

  ANNOTATION: Maximum Cartesian extension of wave.

- const double & **delxyz** = _delxyz

  ANNOTATION: Cartesian displacement.

- const double & **vknmx** = _vknmx

  ANNOTATION: Maximum wave number.

- const double & **delk** = _delk

  ANNOTATION: Wave number variation.

- const double & **delks** = _delks

  ANNOTATION: Squared wave number variation.

- const int & **nlmmt** = _nlmmt

  NLMMT = LM ∗ (LM + 2) ∗ 2.

- const int & **nrvc** = _nrvc

  Read-only view on the number of radial vector coordinates.

Protected Member Functions

- void write_hdf5 (const std::string &file_name)

  Save a Swap2 instance to a HDF5 binary file.

- void write_legacy (const std::string &file_name)

  Save a Swap2 instance to a legacy binary file.

Static Protected Member Functions

- static Swap2 ∗ from_hdf5 (const std::string &file_name)

  Load a Swap2 instance from a HDF5 binary file.

- static Swap2 ∗ from_legacy (const std::string &file_name)

  Load a Swap2 instance from a legacy binary file.

Protected Attributes

- int **_last_vector**

  Index of the last vector element to be filled.
- int **_last_matrix**

  Index of the last matrix element to be filled.
- int **_nkv**

  Number of beam description wave numbers.
- double **_apfafa**

  ANNOTATION: normalized aperture.
- double **_pmf**

  ANNOTATION: aperture correction.
- double **_spd**

  ANNOTATION: cover slip thickness.
- double **_rir**

  ANNOTATION: external over oil refractive index ratio.
- double **_ftcn**

  ANNOTATION: refraction factor.
- double **_fshmx**

  ANNOTATION: cover correction factor.
- double **_vxyzmx**

  ANNOTATION: maximum Cartesian extension of wave.
- double **_delxyz**

  ANNOTATION: Cartesian displacement.
- double **_vknmx**

  ANNOTATION: maximum wave number.
- double **_delk**

  Wave number grid spacing.
- double **_delks**

  Square of wave number grid spacing.
- int **_nlmmt**

  NLMMT = LM ∗ (LM + 2) ∗ 2.
- int **_nrvc**

  Number of radial vector coordinates.

8.28.1 Constructor & Destructor Documentation

8.28.1.1 Swap2()

```
Swap2::Swap2 (
            int nkv )
```

Parameters

nkv	`int` Number of beam description wave numbers.

8.28.2 Member Function Documentation

8.28.2.1 from_binary()

```
Swap2 * Swap2::from_binary (
            const std::string & file_name,
            const std::string & mode = "LEGACY" )  [static]
```

Parameters

file_name	`string` Name of the file.
mode	`string` Format of the file (can be either "HDF5" or "LGEACY". Default is "LEGACY").

Returns

instance: `Swap2 *` Pointer to a newly created Swap2 instance.

8.28.2.2 from_hdf5()

```
Swap2 * Swap2::from_hdf5 (
            const std::string & file_name )  [static], [protected]
```

Parameters

file_name	`string` Name of the file to be loaded.

Returns

instance: `Swap2 *` Pointer to a new Swap2 instance.

8.28.2.3 from_legacy()

```
Swap2 * Swap2::from_legacy (
            const std::string & file_name )  [static], [protected]
```

Parameters

file_name	`string` Name of the file to be loaded.

Returns

instance: `Swap2 *` Pointer to a new Swap2 instance.

8.28.2.4 get_size()

```
long Swap2::get_size (
            int nkv )  [static]
```

Parameters

nkv	`int` Number of beam description wave numbers.

Returns

size: `long` The necessary memory size in bytes.

8.28.2.5 get_vector()

```
double * Swap2::get_vector ( )  [inline]
```

Returns

value: `double *` Pointer to the VKV vector.

8.28.2.6 operator==()

```
bool Swap2::operator== (
            Swap2 & other )
```

Parameters

other	`Swap1 &` Reference to the instance to be compared with.

Returns

result: `bool` True, if the two instances are equal, false otherwise.

8.28.2.7 push_matrix()

```
void Swap2::push_matrix (
            double value )
```

Parameters

value	`double` The value to be pushed in the matrix.

8.28.2.8 push_vector()

```
void Swap2::push_vector (
            double value )  [inline]
```

Parameters

value	`double` The value to be pushed in the vector.

8.28.2.9 set_param()

```
void Swap2::set_param (
            const std::string & param_name,
            double value )
```

Parameters

param_name	`string` Name of the parameter.
value	`double` The value of the parameter.

8.28.2.10 write_binary()

```
void Swap2::write_binary (
            const std::string & file_name,
            const std::string & mode = "LEGACY" )
```

Parameters

file_name	`string` Name of the file.
mode	`string` Format of the file (can be either "HDF5" or "LGEACY". Default is "LEGACY").

8.28.2.11 write_hdf5()

```
void Swap2::write_hdf5 (
            const std::string & file_name )  [protected]
```

Parameters

file_name	`string` Name of the file to be written.

8.28.2.12 write_legacy()

```
void Swap2::write_legacy (
            const std::string & file_name )  [protected]
```

Parameters

file_name	`string` Name of the file to be written.

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

- np_tmcode/src/include/tfrfme.h
- np_tmcode/src/libnptm/tfrfme.cpp

8.29 TFRFME Class Reference

Class to represent the trapping configuration.

`#include <tfrfme.h>`

Public Member Functions

- TFRFME (int lmode, int lm, int nkv, int nxv, int nyv, int nzv)

  Trapping configuration instance constructor.
- ~**TFRFME** ()

  Trapping configuration instance destroyer.
- double * get_x ()

  Get the pointer to the X coordinates vector.
- double * get_y ()

  Get the pointer to the Y coordinates vector.
- double * get_z ()

  Get the pointer to the Z coordinates vector.
- void set_param (const std::string &param_name, double value)

  Set a configuration parameter.
- void write_binary (const std::string &file_name, const std::string &mode="LEGACY")

  Write a trapping configuration instance to binary file.
- bool operator== (const TFRFME &other)

  Test whether two instances of configuration are equal.

Static Public Member Functions

- static TFRFME * from_binary (const std::string &file_name, const std::string &mode="LEGACY")

  Load a trapping configuration instance from binary file.
- static long get_size (int lm, int nkv, int nxv, int nyv, int nzv)

  Calculate the necessary amount of memory to create a new instance.

Public Attributes

- const int & **nlmmt** = _nlmmt

  Read-only view on NLMMT.
- const int & **nrvc** = _nrvc

  Read-only view on NRVC.
- const int & **lmode** = _lmode

  Read-only view on field expansion mode identifier.
- const int & **lm** = _lm

  Read-only view on maximum field expansion order.
- const int & **nkv** = _nkv

  ANNOTATION: Number of wave vectors.
- const int & **nxv** = _nxv

  Read-only view on number of computed X coordinates.
- const int & **nyv** = _nyv

  Read-only view on number of computed Y coordinates.
- const int & **nzv** = _nzv

  Read-only view on number of computed Z coordinates.
- const double & **vk** = _vk

  Read-only view on vacuum wave number.
- const double & **exri** = _exri

  Read-only view on external medium refractive index.
- const double & **an** = _an

  Read-only view on numeric aperture.
- const double & **ff** = _ff

  Read-only view on filling factor.
- const double & **tra** = _tra

  Read-only view on lens transmission.
- const double & **spd** = _spd

  ANNOTATION: Cover slip thickness.
- const double & **frsh** = _frsh

  ANNOTATION: Cover slip correction.
- const double & **exril** = _exril

  ANNOTATION: Oil refractive index.
- dcomplex * **vec_wsum**

  ANNOTATION: Vectorised WSUM matrix.

Protected Member Functions

- void write_hdf5 (const std::string &file_name)

  Save a configuration instance to a HDF5 binary file.
- void write_legacy (const std::string &file_name)

  Save a configuration instance to a legacy binary file.

Static Protected Member Functions

- static TFRFME * from_hdf5 (const std::string &file_name)

  Load a configuration instance from a HDF5 binary file.
- static TFRFME * from_legacy (const std::string &file_name)

  Load a configuration instance from a legacy binary file.

Protected Attributes

- int **_nlmmt**
  NLMMT = 2 ∗ LM ∗ (LM + 2)
- int **_nrvc**
  NRVC = NXV ∗ NYV ∗ NZV.
- int **_lmode**
  Beam description mode.
- int **_lm**
  Maximum field expansion order.
- int **_nkv**
  Number of beam description wave numbers.
- int **_nxv**
  Number of computed X coordinates.
- int **_nyv**
  Number of computed Y coordinates.
- int **_nzv**
  Number of computed Z coordinates.
- double **_vk**
  Vacuum wave number.
- double **_exri**
  External medium refractive index.
- double **_an**
  Numerical aperture.
- double **_ff**
  Filling factor.
- double **_tra**
  Lens transmission.
- double **_spd**
  ANNOTATION: Cover slip thickness.
- double **_frsh**
  ANNOTATION: Cover slip correction factor.
- double **_exril**
  ANNOTATION: Oil refractive index.
- double ∗ **xv**
  Vector of computed x positions.
- double ∗ **yv**
  Vector of computed y positions.
- double ∗ **zv**
  Vector of computed z positions.

8.29.1 Constructor & Destructor Documentation

8.29.1.1 TFRFME()

```
TFRFME::TFRFME (
            int lmode,
            int lm,
            int nkv,
            int nxv,
            int nyv,
            int nzv )
```

Parameters

lmode	`int` Order expansion mode flag.
lm	`int` Maximum field expansion order.
nkv	`int` ANNOTATION: Number of wave vectors.
nxv	`int` Number of computed X coordinates.
nyv	`int` Number of computed Y coordinates.
nzv	`int` Number of computed Z coordinates.

8.29.2 Member Function Documentation

8.29.2.1 from_binary()

```
TFRFME * TFRFME::from_binary (
            const std::string & file_name,
            const std::string & mode = "LEGACY" )  [static]
```

Parameters

file_name	`string` Name of the file.
mode	`string` Format of the file (can be either "HDF5" or "LGEACY". Default is "LEGACY").

Returns

instance: `TFRFME` * Pointer to a newly created configuration instance.

8.29.2.2 from_hdf5()

```
TFRFME * TFRFME::from_hdf5 (
            const std::string & file_name )  [static], [protected]
```

Parameters

file_name	`string` Name of the file to be loaded.

Returns

instance: `TFRFME` * Pointer to a new trapping configuration instance.

8.29.2.3 from_legacy()

```
TFRFME * TFRFME::from_legacy (
            const std::string & file_name )  [static], [protected]
```

Parameters

file_name	`string` Name of the file to be loaded.

Returns

instance: `TFRFME *` Pointer to a new trapping configuration instance.

8.29.2.4 get_size()

```
long TFRFME::get_size (
            int lm,
            int nkv,
            int nxv,
            int nyv,
            int nzv )  [static]
```

Parameters

lm	`int` Maximum field expansion order.
nkv	`int` ANNOTATION: Number of wave vectors.
nxv	`int` Number of computed X coordinates.
nyv	`int` Number of computed Y coordinates.
nzv	`int` Number of computed Z coordinates.

Returns

size: `long` The necessary memory size in bytes.

8.29.2.5 get_x()

```
double * TFRFME::get_x ( )  [inline]
```

Returns

x: `double *` Pointer to X coordinates vector.

8.29.2.6 get_y()

```
double * TFRFME::get_y ( )  [inline]
```

Returns

y: `double *` Pointer to Y coordinates vector.

8.29.2.7 get_z()

```
double * TFRFME::get_z ( )  [inline]
```

Returns

z: `double *` Pointer to Z coordinates vector.

8.29.2.8 operator==()

```
bool TFRFME::operator== (
            const TFRFME & other )
```

Parameters

other	`TFRFME &` Reference to the instance to be compared with.

Returns

result: `bool` True, if the two instances are equal, false otherwise.

8.29.2.9 set_param()

```
void TFRFME::set_param (
            const std::string & param_name,
            double value )
```

Parameters

param_name	`string` Name of the parameter.
value	`double` Value to be stored as parameter.

8.29.2.10 write_binary()

```
void TFRFME::write_binary (
            const std::string & file_name,
            const std::string & mode = "LEGACY" )
```

Parameters

file_name	`string` Name of the file.
mode	`string` Format of the file (can be either "HDF5" or "LGEACY". Default is "LEGACY").

8.29.2.11 write_hdf5()

```
void TFRFME::write_hdf5 (
            const std::string & file_name )  [protected]
```

Parameters

file_name	`string` Name of the file to be written.

8.29.2.12 write_legacy()

```
void TFRFME::write_legacy (
            const std::string & file_name )  [protected]
```

Parameters

file_name	`string` Name of the file to be written.

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

- np_tmcode/src/include/tfrfme.h
- np_tmcode/src/libnptm/tfrfme.cpp

8.30 TransitionMatrix Class Reference

Class to represent the Transition Matrix.

`#include <TransitionMatrix.h>`

Public Member Functions

- TransitionMatrix (int is, int lm, double vk, double exri, dcomplex *_elems, double radius=0.0)
  Default Transition Matrix instance constructor.
- TransitionMatrix (int lm, double vk, double exri, dcomplex **rmi, dcomplex **rei, double radius)
  Transition Matrix instance constructor for single sphere.
- TransitionMatrix (int nlemt, int lm, double vk, double exri, dcomplex **am0m)
  Transition Matrix instance constructor for a cluster of spheres.
- ~**TransitionMatrix** ()
  Transition Matrix instance destroyer.
- void write_binary (const std::string &file_name, const std::string &mode="LEGACY")
  Write the Transition Matrix to a binary file.
- bool operator== (TransitionMatrix &other)
  Test whether two instances of TransitionMatrix are equal.

Static Public Member Functions

- static TransitionMatrix * from_binary (const std::string &file_name, const std::string &mode="LEGACY")

  Build transition matrix from binary input file.
- static void write_binary (const std::string &file_name, np_int nlemt, int lm, double vk, double exri, dcomplex **am0m, const std::string &mode="LEGACY")

  Write a cluster Transition Matrix to a binary file without instanciating it.
- static void write_binary (const std::string &file_name, int lm, double vk, double exri, dcomplex **rmi, dcomplex **rei, double sphere_radius, const std::string &mode="LEGACY")

  Write a single sphere Transition Matrix to a binary file without instanciating it.

Public Attributes

- const int & **is** = _is

  Read-only view on matrix type identifier.
- const int & **l_max** = _l_max

  Read-only view on maximum field expansion order.
- const double & **vk** = _vk

  Read-only view on wave number in scale units.
- const double & **exri** = _exri

  Read-only view on external medium refractive index.
- dcomplex * **elements**

  Vectorized matrix elements.
- const double & **sphere_radius** = _sphere_radius

  Read-only view on sphere radius.
- int * **shape**

  Matrix shape.

Protected Member Functions

- void write_hdf5 (const std::string &file_name)

  Write the Transition Matrix to HDF5 binary output.
- void write_legacy (const std::string &file_name)

  Write the Transition Matrix to legacy binary output.

Static Protected Member Functions

- static TransitionMatrix * from_hdf5 (const std::string &file_name)

  Build transition matrix from a HDF5 binary input file.
- static TransitionMatrix * from_legacy (const std::string &file_name)

  Build transition matrix from a legacy binary input file.
- static void write_hdf5 (const std::string &file_name, np_int nlemt, int lm, double vk, double exri, dcomplex **am0m)

  Write transition matrix data to HDF5 binary output.
- static void write_hdf5 (const std::string &file_name, int lm, double vk, double exri, dcomplex **rmi, dcomplex **rei, double sphere_radius)

  Write transition matrix data to HDF5 binary output.
- static void write_legacy (const std::string &file_name, np_int nlemt, int lm, double vk, double exri, dcomplex **am0m)

  Write transition matrix data to binary output using legacy format.
- static void write_legacy (const std::string &file_name, int lm, double vk, double exri, dcomplex **rmi, dcomplex **rei, double radius)

  Write transition matrix data to binary output using legacy format.

Protected Attributes

- int **_is**

  Matrix type identifier.
- int **_l_max**

  Maximum field expansion order.
- double **_vk**

  Wave number in scale units.
- double **_exri**

  External medium refractive index.
- double **_sphere_radius**

  Sphere radius.

8.30.1 Constructor & Destructor Documentation

8.30.1.1 TransitionMatrix() [1/3]

```
TransitionMatrix::TransitionMatrix (
            int is,
            int lm,
            double vk,
            double exri,
            dcomplex * _elems,
            double radius = 0.0 )
```

Parameters

is	`int` Matrix type identifier
lm	`int` Maximum field expansion order.
vk	`double` Wave number in scale units.
exri	`double` External medium refractive index.
_elems	`complex double *` Vectorized elements of the matrix.
radius	`double` Radius for the single sphere case (defaults to 0.0).

8.30.1.2 TransitionMatrix() [2/3]

```
TransitionMatrix::TransitionMatrix (
            int lm,
            double vk,
            double exri,
            dcomplex ** rmi,
            dcomplex ** rei,
            double radius )
```

This constructor allocates the memory structure needed to represent the transition matrix for the case of a single sphere.

Parameters

lm	`int` Maximum field expansion order.

Parameters

vk	`double` Wave number in scale units.
exri	`double` External medium refractive index.
rmi	`complex double **`
rei	`complex double **`
radius	`double` Radius of the sphere.

8.30.1.3 TransitionMatrix() [3/3]

```
TransitionMatrix::TransitionMatrix (
            int nlemt,
            int lm,
            double vk,
            double exri,
            dcomplex ** am0m )
```

This constructor allocates the memory structure needed to represent the transition matrix for the case of a cluster of spheres.

Parameters

nlemt	`int` Size of the matrix (2 ∗ LE ∗ (LE + 2)).
lm	`int` Maximum field expansion order.
vk	`double` Wave number in scale units.
exri	`double` External medium refractive index.
am0m	`complex double **`

8.30.2 Member Function Documentation

8.30.2.1 from_binary()

```
TransitionMatrix * TransitionMatrix::from_binary (
            const std::string & file_name,
            const std::string & mode = "LEGACY" )  [static]
```

In some cases, it is necessary to perform calculations starting from a pre-computed transition matrix. If this matrix is not available in memory (e.g. because it was calculated by a different process), but it was saved in a binary file, this function can be used to load it back in memory. The function operates in two modes: "LEGACY", which reads the matrix data from a proprietary binary file, having the same structure as the one used by the original FORTRAN code, and "HDF5", which, instead, reads the data from an input file complying with the HDF5 format.

Parameters

file_name	`string` Name of the binary configuration data file.
mode	`string` Binary encoding. Can be one of ["LEGACY", "HDF5"]. Optional (default is "LEGACY").

Returns

config: `TransitionMatrix *` Pointer to object containing the transition matrix data.

8.30.2.2 from_hdf5()

```
TransitionMatrix * TransitionMatrix::from_hdf5 (
            const std::string & file_name )  [static], [protected]
```

This function takes care of the specific task of building a transition matrix memory data structure from a HDF5 binary input file.

Parameters

file_name	`string` Name of the binary configuration data file.

Returns

config: `TransitionMatrix *` Pointer to object containing the transition matrix data.

8.30.2.3 from_legacy()

```
TransitionMatrix * TransitionMatrix::from_legacy (
            const std::string & file_name )  [static], [protected]
```

Parameters

file_name	`string` Name of the binary configuration data file.

Returns

config: `TransitionMatrix *` Pointer to object containing the transition matrix data.

8.30.2.4 operator==()

```
bool TransitionMatrix::operator== (
            TransitionMatrix & other )
```

Transition matrices can be the result of a calculation or of a file input operation, reading from a previously computed object. The == operator tests for the equivalence of two transition matrices, returning `true` if they are equivalnet, `false` otherwise.

Parameters

other	`TransitionMatrix &` Reference to the instance to be compared with.

Returns

result: `bool` True, if the two instances are equal, false otherwise.

8.30.2.5 write_binary() [1/3]

```
void TransitionMatrix::write_binary (
            const std::string & file_name,
            const std::string & mode = "LEGACY" )
```

This function writes a hard-copy of the transition matrix to an output file, making it available for subsequent processes to reload. The function operates in two modes: "LEGACY", which writes a proprietary binary file, using the same structure of the original FORTRAN code, and "HDF5", which, instead, writes the output to a file using the HDF5 format, thus leaving it available for inspection with external tools.

Parameters

file_name	`string` Name of the file to be written.
mode	`string` Binary encoding. Can be one of ["LEGACY", "HDF5"] . Optional (default is "LEGACY").

8.30.2.6 write_binary() [2/3]

```
void TransitionMatrix::write_binary (
            const std::string & file_name,
            int lm,
            double vk,
            double exri,
            dcomplex ** rmi,
            dcomplex ** rei,
            double sphere_radius,
            const std::string & mode = "LEGACY" )  [static]
```

Transition Matrix data can take a large amount of memory. For such reason, attempts to create TransitionMatrix instances only for writing purposes can create unnecessary resource consumption and computing time to duplicate the data into the output buffer. This function offers output to file as a static method. It takes the arguments of a constructor together with the usual arguments to specify the output file name and format, to write the required data directly to a file, without creating a new TransitionMatrix instance. The implementation works for TransitionMatrix objects built for the single sphere case. It belongs to the public class interface and it calls the proper versions of `write_legacy()` and `write_hdf5()`, depending on the requested output format.

Parameters

file_name	`string` Name of the file to be written.
lm	`int` Maximum field expansion order.
vk	`double` Wave number in scale units.
exri	`double` External medium refractive index.
rmi	`complex double **`
rei	`complex double **`
sphere_radius	`double` Radius of the sphere.
mode	`string` Binary encoding. Can be one of ["LEGACY", "HDF5"] . Optional (default is "LEGACY").

8.30.2.7 write_binary() [3/3]

```
void TransitionMatrix::write_binary (
            const std::string & file_name,
            np_int nlemt,
            int lm,
            double vk,
            double exri,
            dcomplex ** am0m,
            const std::string & mode = "LEGACY" )  [static]
```

Transition Matrix data can take a large amount of memory. For such reason, attempts to create TransitionMatrix instances only for writing purposes can create unnecessary resource consumption and computing time to duplicate the data into the output buffer. This function offers output to file as a static method. It takes the arguments of a constructor together with the usual arguments to specify the output file name and format, to write the required data directly to a file, without creating a new TransitionMatrix instance. The implementation works for TransitionMatrix objects built for the CLUSTER case. It belongs to the public class interface and it calls the proper versions of `write_legacy()` and `write_hdf5()`, depending on the requested output format.

Parameters

file_name	`string` Name of the file to be written.
nlemt	`np_int` Size of the matrix (2 ∗ LE ∗ (LE + 2)).
lm	`int` Maximum field expansion order.
vk	`double` Wave number in scale units.
exri	`double` External medium refractive index.
am0m	`complex double **`
mode	`string` Binary encoding. Can be one of ["LEGACY", "HDF5"] . Optional (default is "LEGACY").

8.30.2.8 write_hdf5() [1/3]

```
void TransitionMatrix::write_hdf5 (
            const std::string & file_name )  [protected]
```

This function takes care of the specific task of writing the transition matrix memory data structure to a binary output file formatted according to the HDF5 standard.

Parameters

file_name	`string` Name of the binary configuration data file.

8.30.2.9 write_hdf5() [2/3]

```
void TransitionMatrix::write_hdf5 (
            const std::string & file_name,
            int lm,
            double vk,
            double exri,
            dcomplex ** rmi,
```

```
            dcomplex ** rei,
            double sphere_radius )  [static], [protected]
```

This function takes care of the specific task of writing the transition matrix memory data structure to a binary output file formatted according to the HDF5 standard without a pre-existing instance. It is designed to work for the case of a single sphere.

Parameters

file_name	`string` Name of the binary configuration data file.
lm	`int` Maximum field expansion order.
vk	`double` Wave number in scale units.
exri	`double` External medium refractive index.
rmi	`complex double **`
rei	`complex double **`
sphere_radius	`double` Radius of the sphere.

8.30.2.10 write_hdf5() [3/3]

```
void TransitionMatrix::write_hdf5 (
            const std::string & file_name,
            np_int nlemt,
            int lm,
            double vk,
            double exri,
            dcomplex ** am0m )  [static], [protected]
```

This function takes care of the specific task of writing the transition matrix memory data structure to a binary output file formatted according to the HDF5 standard without a pre-existing instance. It is designed to work for the case of a cluster of spheres.

Parameters

file_name	`string` Name of the binary configuration data file.
nlemt	`np_int` Size of the matrix (2 ∗ LE ∗ (LE + 2)).
lm	`int` Maximum field expansion order.
vk	`double` Wave number in scale units.
exri	`double` External medium refractive index.
am0m	`complex double **`

8.30.2.11 write_legacy() [1/3]

```
void TransitionMatrix::write_legacy (
            const std::string & file_name )  [protected]
```

Parameters

file_name	`string` Name of the binary configuration data file.

8.30.2.12 write_legacy() [2/3]

```
void TransitionMatrix::write_legacy (
            const std::string & file_name,
            int lm,
            double vk,
            double exri,
            dcomplex ** rmi,
            dcomplex ** rei,
            double radius )  [static], [protected]
```

This function takes care of the specific task of writing the transition matrix memory data structure to a binary output file formatted according to the original code structure without a pre-existing instance. It is designed to work for the case of a single sphere.

Parameters

file_name	`string` Name of the binary configuration data file.
lm	`int` Maximum field expansion order.
vk	`double` Wave number in scale units.
exri	`double` External medium refractive index.
rmi	`complex double **`
rei	`complex double **`
radius	`double` Radius of the sphere.

8.30.2.13 write_legacy() [3/3]

```
void TransitionMatrix::write_legacy (
            const std::string & file_name,
            np_int nlemt,
            int lm,
            double vk,
            double exri,
            dcomplex ** am0m )  [static], [protected]
```

This function takes care of the specific task of writing the transition matrix memory data structure to a binary output file formatted according to the format used by the legacy FORTRAN code. It is designed to work for the case of clusters of spheres.

Parameters

file_name	`string` Name of the binary configuration data file.
nlemt	`np_int` Size of the matrix (2 ∗ LE ∗ (LE + 2)).
lm	`int` Maximum field expansion order.
vk	`double` Wave number in scale units.
exri	`double` External medium refractive index.
am0m	`complex double **`

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

- np_tmcode/src/include/TransitionMatrix.h
- np_tmcode/src/libnptm/TransitionMatrix.cpp

8.31 TrappingOutputInfo Class Reference

Class to collect output information for particle trapping.

`#include <outputs.h>`

Public Member Functions

- TrappingOutputInfo (int ift, int iss, int itw, int nx, int ny, int nz)
  TrappingOutputInfo default instance constructor.
- ~**TrappingOutputInfo** ()
  TrappingOutputInfo instance destroyer.
- long get_size ()
  Get the size of a TrappingOutputInfo instance in bytes.
- void set_param (const std::string &pname, double pvalue)
  Set the value of a parameter by its name.
- int write (const std::string &output, const std::string &format)
  Write the output to a file.

Static Public Member Functions

- static long get_size (int ift, int iss, int nx, int ny, int nz)
  Estimate the size of the structure that would be built for given input.

Public Attributes

- const int & **jft** = _jft
  Read-only view on force / torque mode.
- const int & **jss** = _jss
  TWS / TFRFME switch.
- const int & **jtw** = _jtw
  Flag for formatted output.
- const int & **nkv** = _nkv
  Read-only view on number of wave vectors used to describe the radiation field.
- const int & **nxv** = _nxv
  Read-only view on number of vertices along X-axis.
- const int & **nyv** = _nyv
  Read-only view on number of vertices along Y-axis.
- const int & **nzv** = _nzv
  Read-only view on number of vertices along Z-axis.
- const int & **lmode** = _lmode
  Laser mode flag.
- const int & **le** = _le
  Maximum field expansion order for the T-matrix.
- const double & **vk** = _vk
  Vacuum wave number of laser radiation field.
- const double & **exri** = _exri
  Refractive index of the external medium.
- const double & **an** = _an

Numeric aperturte of focusing lens.

- const double & **ff** = _ff

  Lens filling factor.

- const double & **tra** = _tra

  Lens transmittance.

- const double & **spd** = _spd

  Offset of cover slip.

- const double & **frsh** = _frsh

  Cover slip offset normalization.

- const double & **exril** = _exril

  Refractive index of cover lens.

- double * **vec_x**

  Vector of X-coordinates grid spacings.

- double * **vec_y**

  Vector of Y-coordinates grid spacings.

- double * **vec_z**

  Vector of Z-coordinates grid spacings.

- double * **vec_csf1**

  Vector of first components for force cross-sections.

- double * **vec_csf2**

  Vector of second components for force cross-sections.

- double * **vec_csf3**

  Vector of third components for force cross-sections.

- double * **vec_cst1**

  Vector of first components for torque cross-sections.

- double * **vec_cst2**

  Vector of second components for torque cross-sections.

- double * **vec_cst3**

  Vector of third components for torque cross-sections.

Protected Member Functions

- int write_hdf5 (const std::string &file_name)

  Write the output to a HDF5 file.

- int write_legacy (const std::string &output)

  Write the output to a legacy text file.

Protected Attributes

- int **_jft**

  Force / torque mode.

- int **_jss**

  TWS / TFRFME switch.

- int **_jtw**

  Flag for formatted output.

- int **_nkv**

  Number of wave vectors used to describe the radiation field.

- int **_nxv**

  Number of vertices along X-axis.

- int **_nyv**

Number of vertices along Y-axis.

- int **_nzv**

  Number of vertices along Z-axis.

- int **_lmode**

  Laser mode flag.

- int **_le**

  Maximum field expansion order for the T-matrix.

- double **_vk**

  Vacuum wave number of laser radiation field.

- double **_exri**

  Refractive index of the external medium.

- double **_an**

  Numeric aperturte of focusing lens.

- double **_ff**

  Lens filling factor.

- double **_tra**

  Lens transmittance.

- double **_spd**

  Offset of cover slip.

- double **_frsh**

  Cover slip offset normalization.

- double **_exril**

  Refractive index of cover lens.

8.31.1 Detailed Description

The results of the calculation can be saved in different formats. It is therefore convenient to have a proper memory structure that allows for storing the results and flushing them in any of the permitted formats with just one operation. The purpose of the `TrappingOutputInfo` class is to provide a wrapper for the output of the particle trapping solver.

8.31.2 Constructor & Destructor Documentation

8.31.2.1 TrappingOutputInfo()

```
TrappingOutputInfo::TrappingOutputInfo (
            int ift,
            int iss,
            int itw,
            int nx,
            int ny,
            int nz )
```

Parameters

ift	'int' Force / torque toggle mode.
iss	'int' TWS / TFRFME switch.
itw	'int' Formatted output switch (0 - disabled, 1 - enabled).
nx	'int' Number of vertices along the X-axis.
ny	'int' Number of vertices along the Y-axis.
nz	'int' Number of vertices along the Z-axis.

8.31.3 Member Function Documentation

8.31.3.1 get_size() [1/2]

```
long TrappingOutputInfo::get_size ( )
```

Returns

size: long Estimated instance size in bytes.

8.31.3.2 get_size() [2/2]

```
long TrappingOutputInfo::get_size (
            int ift,
            int iss,
            int nx,
            int ny,
            int nz )  [static]
```

Parameters

ift	'int' Force / torque toggle mode.
iss	'int' TWS / TFRFME switch.
nx	'int' Number of vertices along the X-axis.
ny	'int' Number of vertices along the Y-axis.
nz	'int' Number of vertices along the Z-axis.

Returns

size: long Estimated instance size in bytes.

8.31.3.3 set_param()

```
void TrappingOutputInfo::set_param (
            const std::string & pname,
            double pvalue )
```

Parameters

pname	'const string&' Name of the parameter to be set.
pvalue	'double' Value of the parameter.

8.31.3.4 write()

```
int TrappingOutputInfo::write (
            const std::string & output,
            const std::string & format )
```

Parameters

output	`const string &` Path to the output to be written.
format	`const string &` Output format (one of LEGACY or HDF5).

Returns

result: `int` Exit code (0 if successful).

8.31.3.5 write_hdf5()

```
int TrappingOutputInfo::write_hdf5 (
            const std::string & file_name )  [protected]
```

Parameters

file_name	`const string &` Path to the output to be written.

Returns

result: `int` Exit code (0 if successful).

8.31.3.6 write_legacy()

```
int TrappingOutputInfo::write_legacy (
            const std::string & output )  [protected]
```

This function takes care of writing the output using the legacy formatted ASCII structure. If the output file does not exist, it is created. If it exists, the new content is overwritten.

Parameters

output	`const string &` Path to the output to be written.

Returns

result: `int` Exit code (0 if successful).

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

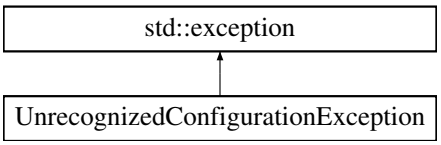
- np_tmcode/src/include/outputs.h
- np_tmcode/src/libnptm/outputs.cpp

8.32 UnrecognizedConfigurationException Class Reference

Exception for unrecognized configuration data sets.

`#include <errors.h>`

Inheritance diagram for UnrecognizedConfigurationException:



Public Member Functions

- UnrecognizedConfigurationException (const std::string &problem)

  Exception instance constructor.
- virtual const char * **what** () const throw ()

  Exception message.

Protected Attributes

- std::string **message**

  Description of the problem.

8.32.1 Constructor & Destructor Documentation

8.32.1.1 UnrecognizedConfigurationException()

```
UnrecognizedConfigurationException::UnrecognizedConfigurationException (
            const std::string & problem )  [inline]
```

Parameters

problem	`string` Description of the problem that occurred.

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

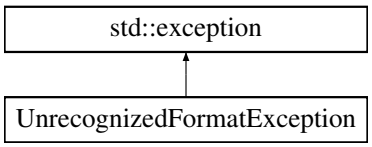
- np_tmcode/src/include/errors.h

8.33 UnrecognizedFormatException Class Reference

Exception for unrecognized file formats.

`#include <errors.h>`

Inheritance diagram for UnrecognizedFormatException:



Public Member Functions

- UnrecognizedFormatException (const std::string &problem)
  Exception instance constructor.
- virtual const char * **what** () const throw ()
  Exception message.

Protected Attributes

- std::string **message**
  Description of the problem.

8.33.1 Constructor & Destructor Documentation

8.33.1.1 UnrecognizedFormatException()

```
UnrecognizedFormatException::UnrecognizedFormatException (
            const std::string & problem )  [inline]
```

Parameters

problem	`string` Description of the problem that occurred.

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

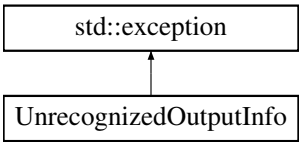
- np_tmcode/src/include/errors.h

8.34 UnrecognizedOutputInfo Class Reference

Exception for wrong OutputInfo NULL calls.

`#include <errors.h>`

Inheritance diagram for UnrecognizedOutputInfo:



Public Member Functions

- UnrecognizedOutputInfo (int requested)
  Exception instance constructor.
- virtual const char * **what** () const throw ()
  Exception message.

Protected Attributes

- std::string **message**

  Description of the problem.

8.34.1 Constructor & Destructor Documentation

8.34.1.1 UnrecognizedOutputInfo()

```
UnrecognizedOutputInfo::UnrecognizedOutputInfo (
            int requested )  [inline]
```

Parameters

requested	`int` The index that was requested.

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

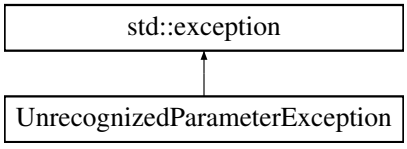
- np_tmcode/src/include/errors.h

8.35 UnrecognizedParameterException Class Reference

Exception for unrecognized parameters.

`#include <errors.h>`

Inheritance diagram for UnrecognizedParameterException:



Public Member Functions

- UnrecognizedParameterException (const std::string &problem)

  Exception instance constructor.
- virtual const char * **what** () const throw ()

  Exception message.

Protected Attributes

- std::string **message**

  Description of the problem.

8.35.1 Constructor & Destructor Documentation

8.35.1.1 UnrecognizedParameterException()

```
UnrecognizedParameterException::UnrecognizedParameterException (
            const std::string & problem )  [inline]
```

Parameters

problem	`string` Description of the problem that occurred.

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

- np_tmcode/src/include/errors.h

8.36 VirtualAsciiFile Class Reference

Virtual representation of an ASCII file.

`#include <file_io.h>`

Public Member Functions

- VirtualAsciiFile (int32_t lines=0)
  VirtualAsciiFile instance constructor.
- VirtualAsciiFile (const VirtualAsciiFile &rhs)
  VirtualAsciiFile copy constructor.
- VirtualAsciiFile (const mixMPI *mpidata, int rr)
  VirtualAsciiFile instance constructor copying all contents off MPISend() calls from MPI process rr.
- ~**VirtualAsciiFile** ()
  VirtualAsciiFile instance destroyer.
- void append (VirtualAsciiFile &rhs)
  Append another VirtualAsciiFile at the end of the current instance.
- void append_line (const std::string &line)
  Append a line at the end of the file.
- int append_to_disk (const std::string &file_name)
  Append the contents of the VirtualAsciiFile to a physical file on disk.
- int insert (int32_t position, VirtualAsciiFile &rhs, int32_t start=0, int32_t end=0)
  Insert another VirtualAsciiFile at a given position.
- int32_t number_of_lines ()
  Get the number of lines in the current instance.
- int write_to_disk (const std::string &file_name)
  Write virtual file contents to a real file on disk.
- void mpisend (const mixMPI *mpidata)
  Send VirtualAsciiFile instance to MPI process 0 via MPISend() calls.

Protected Attributes

- std::vector< std::string > * _file_lines
  The number of lines.

8.36.1 Constructor & Destructor Documentation

8.36.1.1 VirtualAsciiFile() [1/3]

```
VirtualAsciiFile::VirtualAsciiFile (
            int32_t lines = 0 )
```

Parameters

lines	`int32_t` Number of lines, if known in advance (optional, default is 0).

8.36.1.2 VirtualAsciiFile() [2/3]

```
VirtualAsciiFile::VirtualAsciiFile (
            const VirtualAsciiFile & rhs )
```

Parameters

rhs	`const VirtualAsciiFile&` Reference to a VirtualAsciiFile instance.

8.36.1.3 VirtualAsciiFile() [3/3]

```
VirtualAsciiFile::VirtualAsciiFile (
            const mixMPI * mpidata,
            int rr )
```

Parameters

mpidata	`mixMPI *` pointer to MPI data structure.
rr	`int` rank of the MPI process sending the data.

8.36.2 Member Function Documentation

8.36.2.1 append()

```
void VirtualAsciiFile::append (
            VirtualAsciiFile & rhs )
```

Parameters

rhs	`const VirtualAsciiFile&` Reference to the VirtualAsciiFile to be appended.

8.36.2.2 append_line()

```
void VirtualAsciiFile::append_line (
            const std::string & line )
```

Parameters

line	`const string&` Reference to a string representing the line.

8.36.2.3 append_to_disk()

```
int VirtualAsciiFile::append_to_disk (
            const std::string & file_name )
```

Parameters

file_name	`const string&` Name of the file to append contents to.

Returns

result: `int` A result code (0 if successful).

8.36.2.4 insert()

```
int VirtualAsciiFile::insert (
            int32_t position,
            VirtualAsciiFile & rhs,
            int32_t start = 0,
            int32_t end = 0 )
```

This function inserts a target VirtualAsciiFile in the current one at the given position. Optionally, a range of lines to be inserted can be specified, otherwise the full content of the target file is inserted. This function DOES NOT increase the size of the inner storage and it can only be used if the inner storage has already been adjusted to contain the insertion target.

Parameters

position	`int32_t` The position at which the other file is inserted in this one.
rhs	`const VirtualAsciiFile&` The refence to the VirtualAsciiFile to be inserted.
start	`int32_t` The first line to be inserted (optional, default is 0).
end	`int32_t` The last line to be inserted (optional, default is 0 to read all).

Returns

result: `int` A result code (0 if successful).

8.36.2.5 mpisend()

```
void VirtualAsciiFile::mpisend (
            const mixMPI * mpidata )
```

Parameters

mpidata	`mixMPI *` pointer to MPI data structure.

8.36.2.6 number_of_lines()

```
int32_t VirtualAsciiFile::number_of_lines ( )  [inline]
```

Returns

size: `int32_t` The number of lines in the VirtualAsciiFile instance.

8.36.2.7 write_to_disk()

```
int VirtualAsciiFile::write_to_disk (
            const std::string & file_name )
```

Parameters

file_name	`const string&` Name of the file to append contents to.

Returns

result: `int` A result code (0 if successful).

8.36.3 Member Data Documentation

8.36.3.1 _file_lines

```
std::vector<std::string>* VirtualAsciiFile::_file_lines  [protected]
```

A vector of strings representing the file lines.

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

- np_tmcode/src/include/file_io.h
- np_tmcode/src/libnptm/file_io.cpp

8.37 VirtualBinaryFile Class Reference

Virtual representation of a binary file.

```
#include <file_io.h>
```

Public Member Functions

- **VirtualBinaryFile** ()

  VirtualBinaryFile empty instance constructor.
- VirtualBinaryFile (const VirtualBinaryFile &rhs)

  VirtualBinaryFile copy constructor.
- VirtualBinaryFile (const mixMPI *mpidata, int rr)

  VirtualBinaryFile instance constructor copying all contents off MPISend() calls from MPI process rr.
- ~**VirtualBinaryFile** ()

  VirtualBinaryFile instance destroyer.
- void append (VirtualBinaryFile &rhs)

  Append another VirtualBinaryFile at the end of the current instance.
- void append_line (const VirtualBinaryLine &line)

  Append a line at the end of the file.
- int append_to_disk (const std::string &file_name)

  Append the contents of the VirtualBinaryFile to a physical file on disk.
- int32_t number_of_lines ()

  Get the number of lines in the current instance.
- int write_to_disk (const std::string &file_name)

  Write virtual file contents to a real file on disk.
- void mpisend (const mixMPI *mpidata)

  Send VirtualBinaryFile instance to MPI process 0 via MPISend() calls.

Public Attributes

- std::vector< VirtualBinaryLine > * _file_lines

  The number of lines.

8.37.1 Constructor & Destructor Documentation

8.37.1.1 VirtualBinaryFile() [1/2]

```
VirtualBinaryFile::VirtualBinaryFile (
            const VirtualBinaryFile & rhs )
```

Parameters

rhs	`const VirtualBinaryFile&` Reference to a VirtualBinaryFile instance.

8.37.1.2 VirtualBinaryFile() [2/2]

```
VirtualBinaryFile::VirtualBinaryFile (
            const mixMPI * mpidata,
            int rr )
```

Parameters

mpidata	`mixMPI *` pointer to MPI data structure.
rr	`int` rank of the MPI process sending the data.

8.37.2 Member Function Documentation

8.37.2.1 append()

```
void VirtualBinaryFile::append (
            VirtualBinaryFile & rhs )
```

Parameters

rhs	`const VirtualBinaryFile&` Reference to the VirtualBinaryFile to be appended.

8.37.2.2 append_line()

```
void VirtualBinaryFile::append_line (
            const VirtualBinaryLine & line )
```

Parameters

line	`const string&` Reference to a string representing the line.

8.37.2.3 append_to_disk()

```
int VirtualBinaryFile::append_to_disk (
            const std::string & file_name )
```

Parameters

file_name	`const string&` Name of the file to append contents to.

Returns

result: `int` A result code (0 if successful).

8.37.2.4 mpisend()

```
void VirtualBinaryFile::mpisend (
            const mixMPI * mpidata )
```

Parameters

mpidata	`mixMPI *` pointer to MPI data structure.

8.37.2.5 number_of_lines()

```
int32_t VirtualBinaryFile::number_of_lines ( )  [inline]
```

Returns

size: `int32_t` The number of lines in the VirtualBinaryFile instance.

8.37.2.6 write_to_disk()

```
int VirtualBinaryFile::write_to_disk (
            const std::string & file_name )
```

Parameters

file_name	`const string&` Name of the file to append contents to.

Returns

result: `int` A result code (0 if successful).

8.37.3 Member Data Documentation

8.37.3.1 _file_lines

```
std::vector<VirtualBinaryLine>* VirtualBinaryFile::_file_lines
```

A vector of strings representing the file lines.

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

- np_tmcode/src/include/file_io.h
- np_tmcode/src/libnptm/file_io.cpp

8.38 VirtualBinaryLine Class Reference

Virtual representation of a binary file line.

```
#include <file_io.h>
```

Public Member Functions

- VirtualBinaryLine (int mydata)
  VirtualBinaryLine instance constructor for `int` data.
- VirtualBinaryLine (long mydata)
  VirtualBinaryLine instance constructor for `long` data.
- VirtualBinaryLine (float mydata)
  VirtualBinaryLine instance constructor for single-precision floating point data.
- VirtualBinaryLine (double mydata)
  VirtualBinaryLine instance constructor for double-precision floating point data.
- VirtualBinaryLine (dcomplex mydata)

VirtualBinaryLine instance constructor for `dcomplex` *data.*

- VirtualBinaryLine (const VirtualBinaryLine &rhs)

  VirtualBinaryLine copy constructor.

- VirtualBinaryLine (const mixMPI *mpidata, int rr)

  VirtualBinaryLine instance constructor copying all contents off MPISend() calls from MPI process rr.

- ~**VirtualBinaryLine** ()

  VirtualBinaryLine instance destroyer.

- void mpisend (const mixMPI *mpidata)

  Send VirtualBinaryLine instance to MPI process 0 via MPISend() calls.

Public Attributes

- char * **_data_pointer**

  *The pointer to the piece of data to be written, cast to char *.*

- size_t **_data_size**

  the size of the data block.

- const char * **data_pointer** = _data_pointer

  Read only view of `_data_pointer`.

- const size_t & **data_size** = _data_size

  Read only view of `_data_size`.

8.38.1 Constructor & Destructor Documentation

8.38.1.1 VirtualBinaryLine() [1/7]

```
VirtualBinaryLine::VirtualBinaryLine (
            int mydata )
```

Parameters

mydata	`int` The piece of data to put in the line.

8.38.1.2 VirtualBinaryLine() [2/7]

```
VirtualBinaryLine::VirtualBinaryLine (
            long mydata )
```

Parameters

mydata	`long` The piece of data to put in the line.

8.38.1.3 VirtualBinaryLine() [3/7]

```
VirtualBinaryLine::VirtualBinaryLine (
            float mydata )
```

Parameters

mydata	`float` The piece of data to put in the line.

8.38.1.4 VirtualBinaryLine() [4/7]

```
VirtualBinaryLine::VirtualBinaryLine (
            double mydata )
```

Parameters

mydata	`double` The piece of data to put in the line.

8.38.1.5 VirtualBinaryLine() [5/7]

```
VirtualBinaryLine::VirtualBinaryLine (
            dcomplex mydata )
```

Parameters

mydata	`dcomplex` The piece of data to put in the line.

8.38.1.6 VirtualBinaryLine() [6/7]

```
VirtualBinaryLine::VirtualBinaryLine (
            const VirtualBinaryLine & rhs )
```

Parameters

rhs	`const VirtualBinaryLine&` Reference to a VirtualBinaryLine instance.

8.38.1.7 VirtualBinaryLine() [7/7]

```
VirtualBinaryLine::VirtualBinaryLine (
            const mixMPI * mpidata,
            int rr )
```

Parameters

mpidata	`mixMPI *` pointer to MPI data structure.
rr	`int` rank of the MPI process sending the data.

8.38.2 Member Function Documentation

8.38.2.1 mpisend()

```
void VirtualBinaryLine::mpisend (
            const mixMPI * mpidata )
```

Parameters

mpidata	`mixMPI` * pointer to MPI data structure.

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

- np_tmcode/src/include/file_io.h
- np_tmcode/src/libnptm/file_io.cpp

Chapter 9

File Documentation

9.1 np_tmcode/src/cluster/cluster.cpp File Reference

Implementation of the calculation for a cluster of spheres.

```
#include <chrono>
#include <cmath>
#include <cstdio>
#include <exception>
#include <fstream>
#include <hdf5.h>
#include <string>
#include "../include/types.h"
#include "../include/errors.h"
#include "../include/logging.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
#include "../include/sph_subs.h"
#include "../include/clu_subs.h"
#include "../include/TransitionMatrix.h"
#include "../include/algebraic.h"
#include "../include/List.h"
#include "../include/file_io.h"
#include "../include/utils.h"
#include "../include/outputs.h"
#include "../include/IterationData.h"
```

Functions

- int cluster_jxi488_cycle (int jxi488, ScattererConfiguration *sconf, GeometryConfiguration *gconf, ScatteringAngles *sa, ClusterIterationData *cid, ClusterOutputInfo *oi, const string &output_path, VirtualBinaryFile *vtppoanp)

  Main calculation loop.
- void cluster (const string &config_file, const string &data_file, const string &output_path, const mixMPI *mpidata)

  C++ implementation of CLU.

9.1.1 Function Documentation

9.1.1.1 cluster()

```
void cluster (
            const string & config_file,
            const string & data_file,
            const string & output_path,
            const mixMPI * mpidata )
```

Parameters

config_file	`const string &` Name of the configuration file.
data_file	`const string &` Name of the input data file.
output_path	`const string &` Name of the directory to write the output files in.
mpidata	`const mixMPI *` Pointer to a mixMPI data structure.

9.1.1.2 cluster_jxi488_cycle()

```
int cluster_jxi488_cycle (
            int jxi488,
            ScattererConfiguration * sconf,
            GeometryConfiguration * gconf,
            ScatteringAngles * sa,
            ClusterIterationData * cid,
            ClusterOutputInfo * oi,
            const string & output_path,
            VirtualBinaryFile * vtppoanp )
```

The solution of the scattering problem for different wavelengths is an embarrasingly parallel task. This function, therefore, collects all the operations that can be independently executed by different processes, after the configuration stage and the first calculation loop have been executed.

Parameters

jxi488	`int` Wavelength loop index.
sconf	`ScattererConfiguration *` Pointer to a `ScattererConfiguration` object.
gconf	`GeometryConfiguration *` Pointer to a `GeometryConfiguration` object.
sa	`ScatteringAngles *` Pointer to a `ScatteringAngles` object.
cid	`ClusterIterationData *` Pointer to a `ClusterIterationData` object.
oi	`ClusterOutputInfo *` Pointer to a `ClusterOutputInfo` object.
output_path	`const string &` Path to the output directory.
vtppoanp	`VirtualBinaryFile *` Pointer to a `VirtualBinaryFile` object.

9.2 np_tmcode/src/cluster/np_cluster.cpp File Reference

Cluster of spheres scattering problem handler.

```
#include <cstdio>
#include <string>
#include "../include/types.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
```

Functions

- void cluster (const string &config_file, const string &data_file, const string &output_path, const mixMPI *mpidata)

  C++ implementation of CLU.
- int main (int argc, char **argv)

  Main program entry point.

9.2.1 Detailed Description

This program emulates the execution work-flow originally handled by the FORTRAN EDFB and CLU codes, which undertook the task of solving the scattering calculation for the case of a cluster of spheres, with one or more materials. The program is designed to work in two modes: emulation and enhanced. The emulation mode is activated by invoking the program without arguments. In this case, the code looks for hard-coded default input and writes output in the execution folder, replicating the behaviour of the original FORTRAN code. Advanced mode, instead, is activated by passing command line arguments that locate the desired input files and a valid folder to write the output into. The emulation mode is useful for testing, while the advanced mode implements the possibility to change input and output options, without having to modify the code.

9.2.2 Function Documentation

9.2.2.1 cluster()

```
void cluster (
            const string & config_file,
            const string & data_file,
            const string & output_path,
            const mixMPI * mpidata )  [extern]
```

Parameters

config_file	const string & Name of the configuration file.
data_file	const string & Name of the input data file.
output_path	const string & Name of the directory to write the output files in.
mpidata	const mixMPI * Pointer to a mixMPI data structure.

9.2.2.2 main()

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

This is the starting point of the execution flow. Here we may choose how to configure the code, e.g. by loading a legacy configuration file or some otherwise formatted configuration data set. The code can be linked to a luncher script or to a GUI oriented application that performs the configuration and runs the main program.

Parameters

argc	`int` The number of arguments given in command-line.
argv	`char **` The vector of command-line arguments.

Returns

result: `int` An exit code passed to the OS (0 for succesful execution).

9.3 np_tmcode/src/include/algebraic.h File Reference

Declaration of algebraic functions with different call-backs.

Functions

- void invert_matrix (dcomplex **mat, np_int size, int &ier, int &maxrefiters, double &accuracygoal, int refinemode, const string &output_path, int jxi488, np_int max_size=0, int target_device=0)

  Perform in-place matrix inversion.

9.3.1 Detailed Description

In principle, the system that runs NP_TMcode may offer various types of optimized features, such as multi-core or multi-node scaling, GPU offload, or external libraries. This header collects a set of functions that can perform standard algebraic operations choosing the most optimized available system as a call-back. If no optimization is detected, eventually the legacy serial function implementation is used as a fall-back.

9.3.2 Function Documentation

9.3.2.1 invert_matrix()

```
void invert_matrix (
            dcomplex ** mat,
            np_int size,
            int & ier,
            int & maxrefiters,
            double & accuracygoal,
            int refinemode,
            const string & output_path,
            int jxi488,
            np_int max_size = 0,
            int target_device = 0 )
```

Parameters

mat	`complex double **` The matrix to be inverted (must be a square matrix).
size	`np_int` The size of the matrix (i.e. the number of its rows or columns).
ier	`int &` Reference to an integer variable for returning a result flag.
maxrefiters	`int &` Reference to the maximum number of refinement iterations.
accuracygoal	`double &` Reference to the requested accuracy level.
refinemode	`int` Flag for refinement mode selection.
output_path	`const string &` Path where the output needs to be placed.
jxi488	`int` Index of the current wavelength calculation.
max_size	`np_int` The maximum expected size (required by some call-backs, optional, defaults to 0).
target_device	`int` ID of target GPU, if available (defaults to 0).

9.4 algebraic.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00029 #ifndef INCLUDE_ALGEBRAIC_H_
00030 #define INCLUDE_ALGEBRAIC_H_
00031
00032 using namespace std;
00033
00047 void invert_matrix(dcomplex **mat, np_int size, int &ier, int &maxrefiters, double &accuracygoal, int
       refinemode, const string& output_path, int jxi488, np_int max_size=0, int target_device=0);
00048
00049 #endif
```

9.5 np_tmcode/src/include/clu_subs.h File Reference

C++ porting of CLU functions and subroutines.

Functions

- void apc (double ****zpv, int le, dcomplex **am0m, dcomplex **w, double sqk, double **gapr, dcomplex **gapp)

  Compute the asymmetry-corrected scattering cross-section of a cluster.
- void apcra (double ****zpv, const int le, dcomplex **am0m, int inpol, double sqk, double **gaprm, dcomplex **gappm)

  Compute the asymmetry-corrected scattering cross-section under random average conditions.
- dcomplex cdtp (dcomplex z, dcomplex *vec_am, int i, int jf, int k, int nj, np_int istep)

  Complex inner product.

- double cgev (int ipamo, int mu, int l, int m)

  C++ porting of CGEV. ANNOTATION: Get weight of T-matrix element.
- void cms (dcomplex **am, ParticleDescriptor *c1)

  Build the multi-centered M-matrix of the cluster.
- void crsm1 (double vk, double exri, ParticleDescriptor *c1)

  Compute orientation-averaged scattered field intensity.
- dcomplex ghit_d (int ihi, int ipamo, int nbl, int l1, int m1, int l2, int m2, ParticleDescriptor *c1, double *rac3j)

  Compute the transfer vector from N2 to N1.
- dcomplex ghit (int ihi, int ipamo, int nbl, int l1, int m1, int l2, int m2, ParticleDescriptor *c1)

  Compute the transfer vector from N2 to N1.
- void hjv (double exri, double vk, int &jer, int &lcalc, dcomplex &arg, ParticleDescriptor *c1)

  Compute Hankel funtion and Bessel functions.
- void lucin (dcomplex **am, const np_int nddmst, np_int n, int &ier)

  Invert the multi-centered M-matrix.
- void mextc (double vk, double exri, dcomplex **fsac, double **cextlr, double **cext)

  Compute the average extinction cross-section.
- void pcros (double vk, double exri, ParticleDescriptor *c1)

  Compute cross-sections and forward scattering amplitude for the cluster.
- void pcrsm0 (double vk, double exri, int inpol, ParticleDescriptor *c1)

  Compute orientation-averaged cross-sections and forward scattering amplitude.
- void polar (double x, double y, double z, double &r, double &cth, double &sth, double &cph, double &sph)

  Transform Cartesian quantities to spherical coordinates ones.
- void r3j000 (int j2, int j3, double *rac3j)

  Compute the 3j symbol for Clebsch-Gordan coefficients towards J=0.
- void r3jjr (int j2, int j3, int m2, int m3, double *rac3j)

  Compute the 3j symbol for Clebsch-Gordan coefficients for JJ transitions.
- void r3jjr_d (int j2, int j3, int m2, int m3, double *rac3j)

  Compute the 3j symbol for Clebsch-Gordan coefficients for JJ transitions.
- void r3jmr (int j1, int j2, int j3, int m1, double *rac3j)

  Compute the 3j symbol for Clebsch-Gordan coefficients for JM transitions.
- void raba (int le, dcomplex **am0m, dcomplex **w, double **tqce, dcomplex **tqcpe, double **tqcs, dcomplex **tqcps)

  Compute radiation torques on a particle in Cartesian coordinates.
- void rftr (double *u, double *up, double *un, double *gapv, double extins, double scatts, double &rapr, double &cosav, double &fp, double &fn, double &fk, double &fx, double &fy, double &fz)

  Compute the radiation force Cartesian components.
- void scr0 (double vk, double exri, ParticleDescriptor *c1)

  Compute Mie cross-sections for the sphere units in the cluster.
- void scr2 (double vk, double vkarg, double exri, double *duk, ParticleDescriptor *c1)

  Compute the scattering amplitude for a single sphere in an aggregate.
- void str (double **rcf, ParticleDescriptor *c1)

  Transform sphere Cartesian coordinates to spherical coordinates.
- void tqr (double *u, double *up, double *un, double *tqev, double *tqsv, double &tep, double &ten, double &tek, double &tsp, double &tsn, double &tsk)

  Compute radiation torques on particles on a k-vector oriented system.
- void ztm (dcomplex **am, ParticleDescriptor *c1)

  Calculate the single-centered inversion of the M-matrix.

9.5.1 Detailed Description

This library includes a collection of functions that are used to solve the scattering problem in the case of a cluster of spheres. The functions that were generalized from the case of the single sphere are imported the sph_subs.h library. As it occurs with the single sphere case functions, in most cases, the results of calculations do not fall back to fundamental data types. They are rather multi-component structures. In order to manage access to such variety of return values, most functions are declared as void and they operate on output arguments passed by reference.

9.5.2 Function Documentation

9.5.2.1 apc()

```
void apc (
              double **** zpv,
              int le,
              dcomplex ** am0m,
              dcomplex ** w,
              double sqk,
              double ** gapr,
              dcomplex ** gapp )
```

This function computes the product between the geometrical asymmetry parameter and the scattering cross-section, like aps(), but for a cluster of spheres. See Eq. (3.16) in Borghese, Denti & Saija (2007).

Parameters

zpv	double ****
le	int
am0m	complex double **
w	complex double **
sqk	double
gapr	double **
gapp	complex double **

9.5.2.2 apcra()

```
void apcra (
              double **** zpv,
              const int le,
              dcomplex ** am0m,
              int inpol,
              double sqk,
              double ** gaprm,
              dcomplex ** gappm )
```

This function computes the product between the geometrical asymmetry parameter and the scattering cross-section of a cluster using the random average directions.

Parameters

zpv	double ****

Parameters

le	`int`
am0m	`complex double **`
inpol	`int` Polarization type.
sqk	`double`
gaprm	`double **`
gappm	`complex double **`

9.5.2.3 cdtp()

```
dcomplex cdtp (
            dcomplex z,
            dcomplex * vec_am,
            int i,
            int jf,
            int k,
            int nj,
            np_int istep )
```

This function performs the complex inner product. It is used by `lucin()`.

Parameters

z	`complex double` Starting element.
vec_am	`complex double *` Vectorized matrix.
i	`int` Index of first sub-matrix row.
jf	`int` Index of first submatrix column.
k	`int` Index of second sub-matrix row.
nj	`int` Range of first sub-matrix columns.
istep	`np_int` Size of rows in the matrix.

9.5.2.4 cgev()

```
double cgev (
            int ipamo,
            int mu,
            int l,
            int m )
```

Parameters

ipamo	`int`
mu	`int`
l	`int`
m	`int`

Returns

result: double

9.5.2.5 cms()

```
void cms (
            dcomplex ** am,
            ParticleDescriptor * c1 )
```

This function constructs the multi-centered M-matrix of the cluster, according to Eq. (5.28) of Borghese, Denti & Saija (2007).

Parameters

am	complex double **
c1	ParticleDescriptor *

9.5.2.6 crsm1()

```
void crsm1 (
            double vk,
            double exri,
            ParticleDescriptor * c1 )
```

This function computes the intensity of the scattered field for the cluster, averaged on the orientations. It is invoked for IAVM=1 (ANNOTATION: geometry referred to the meridional plane).

Parameters

vk	double Wave number.
exri	double External medium refractive index.
c1	ParticleDescriptor *

9.5.2.7 ghit()

```
dcomplex ghit (
            int ihi,
            int ipamo,
            int nbl,
            int l1,
            int m1,
            int l2,
            int m2,
            ParticleDescriptor * c1 )
```

This function computes the transfer vector going from N2 to N1, using either Hankel, Bessel or Bessel from origin functions.

Parameters

ihi	int
ipamo	int
nbl	int
l1	int
m1	int
l2	int
m2	int
c1	ParticleDescriptor * Poiunter to a ParticleDescriptor instance.

9.5.2.8 ghit_d()

```
dcomplex ghit_d (
            int ihi,
            int ipamo,
            int nbl,
            int l1,
            int m1,
            int l2,
            int m2,
            ParticleDescriptor * c1,
            double * rac3j )
```

This function computes the transfer vector going from N2 to N1, using either Hankel, Bessel or Bessel from origin functions.

Parameters

ihi	int
ipamo	int
nbl	int
l1	int
m1	int
l2	int
m2	int
c1	ParticleDescriptor *
rac3j	dcomplex *

9.5.2.9 hjv()

```
void hjv (
            double exri,
            double vk,
            int & jer,
            int & lcalc,
            dcomplex & arg,
            ParticleDescriptor * c1 )
```

This function constructs the Hankel function and the Bessel functions vectors. See page 331 in Borghese, Denti & Saija (2007).

Parameters

exri	`double` External medium refractive index.
vk	`double` Wave number.
jer	`int &` Reference to error code flag.
lcalc	`int &` Reference to the highest order accounted for in calculation.
arg	`complex\<double\> &`
c1	`ParticleDescriptor *`

9.5.2.10 lucin()

```
void lucin (
            dcomplex ** am,
            const np_int nddmst,
            np_int n,
            int & ier )
```

This function performs the inversion of the multi-centered M-matrix through LU decomposition. See Eq. (5.29) in Borghese, Denti & Saija (2007).

Parameters

am	`complex double **`
nddmst	`const int64_t`
n	`int64_t`
ier	`int &`

9.5.2.11 mextc()

```
void mextc (
            double vk,
            double exri,
            dcomplex ** fsac,
            double ** cextlr,
            double ** cext )
```

This funbction computes the average extinction cross-section starting from the definition of the scattering amplitude. See Sec. 3.2.1 of Borghese, Denti & Saija (2007).

Parameters

vk	`double` Wave number.
exri	`double` External medium refractive index.
fsac	Matrix of complex
cextlr	`double **`
cext	`double **`

9.5.2.12 pcros()

```
void pcros (
            double vk,
            double exri,
            ParticleDescriptor * c1 )
```

This function computes the scattering, absorption and extinction cross-sections of the cluster, together with the Forward Scattering Amplitude.

This function is intended to evaluate the particle cross-section. QUESTIUON: correct?

Parameters

vk	`double` Wave number.
exri	`double` External medium refractive index.
c1	`ParticleDescriptor *`

9.5.2.13 pcrsm0()

```
void pcrsm0 (
            double vk,
            double exri,
            int inpol,
            ParticleDescriptor * c1 )
```

This function computes the orientation-averaged scattering, absorption and extinction cross-sections of the cluster, together with the averaged Forward Scattering Amplitude.

Parameters

vk	`double` Wave number.
exri	`double` External medium refractive index.
inpol	`int` Incident field polarization type.
c1	`ParticleDescriptor *`

9.5.2.14 polar()

```
void polar (
            double x,
            double y,
            double z,
            double & r,
            double & cth,
            double & sth,
            double & cph,
            double & sph )
```

This function performs a conversion from the Cartesian coordinates system to the spherical one. It is used by `sphar()`.

Parameters

x	`double` X-axis Cartesian coordinate.
y	`double` Y-axis Cartesian coordinate.
z	`double` Z-axis Cartesian coordinate.
r	`double &` Reference to radial vector (output value).
cth	`double &` Reference to the cosine of the azimuth coordinate (output value).
sth	`double &` Reference to the sine of the azimuth coordinate (output value).
cph	`double &` Reference to the cosine of the elevation coordinate (output value).
sph	`double &` Reference to the sine of the elevation coordinate (output value).

9.5.2.15 r3j000()

```
void r3j000 (
            int j2,
            int j3,
            double * rac3j )
```

This function calculates the 3j(J,J2,J3;0,0,0) symbol for the Clebsch-Gordan coefficients. See Appendix a.3.1 in Borghese, Denti & Saija (2007).

Parameters

j2	`int`
j3	`int`
rac3j	`double *` Vector of 3j symbols.

9.5.2.16 r3jjr()

```
void r3jjr (
            int j2,
            int j3,
            int m2,
            int m3,
            double * rac3j )
```

This function calculates the 3j(J,J2,J3;-M2-M3,M2,M3) symbol for the Clebsch-Gordan coefficients. See Appendix a.3.1 in Borghese, Denti & Saija (2007).

Parameters

j2	`int`
j3	`int`
m2	`int`
m3	`int`
rac3j	`double *` Vector of 3j symbols.

9.5.2.17 r3jjr_d()

```
void r3jjr_d (
            int j2,
            int j3,
            int m2,
            int m3,
            double * rac3j )
```

This function calculates the 3j(J,J2,J3;-M2-M3,M2,M3) symbol for the Clebsch-Gordan coefficients. See Appendix a.3.1 in Borghese, Denti & Saija (2007).

Parameters

j2	int
j3	int
m2	int
m3	int
rac3j	double *

9.5.2.18 r3jmr()

```
void r3jmr (
            int j1,
            int j2,
            int j3,
            int m1,
            double * rac3j )
```

This function calculates the 3j(J,J2,J3;M1,M,-M1-M) symbol for the Clebsch-Gordan coefficients. See Appendix a.3.1 in Borghese, Denti & Saija (2007).

Parameters

j1	int
j2	int
j3	int
m1	int
rac3j	double * Vector of 3j symbols.

9.5.2.19 raba()

```
void raba (
            int le,
            dcomplex ** am0m,
            dcomplex ** w,
            double ** tqce,
            dcomplex ** tqcpe,
            double ** tqcs,
            dcomplex ** tqcps )
```

This function computes radiation torque on on a cluster of spheres as the result of the difference between the extinction and the scattering contributions for a Cartesian coordinate system, as `rabas()`. See Sec. 4.9 in Borghese, Denti & Saija (2007).

Parameters

le	`int`
am0m	`complex double **`
w	`complex double **`
tqce	`double **`
tqcpe	`complex double **`
tqcs	`double **`
tqcps	`complex double **`

9.5.2.20 rftr()

```
void rftr (
            double * u,
            double * up,
            double * un,
            double * gapv,
            double extins,
            double scatts,
            double & rapr,
            double & cosav,
            double & fp,
            double & fn,
            double & fk,
            double & fx,
            double & fy,
            double & fz )
```

This function computes the Cartesian components of the radiation force exerted on a particle. See Sec. 3.2.1 in Borghese, Denti & Saija (2007).

Parameters

u	`double *`
up	`double *`
un	`double *`
gapv	`double *`
extins	`double`
scatts	`double`
rapr	`double &`
cosav	`double &`
fp	`double &`
fn	`double &`
fk	`double &`
fx	`double &`
fy	`double &`
fz	`double &`

9.5.2.21 scr0()

```
void scr0 (
            double vk,
            double exri,
            ParticleDescriptor * c1 )
```

This function computes the scattering, absorption and extinction cross-sections for the spheres composing the cluster, in terms of Mie coefficients, together with the Forward Scattering Amplitude. See Sec. 4.2.1 in Borghese, Denti & Saija (2007).

Parameters

vk	`double` Wave number
exri	`double` External medium refractive index.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.5.2.22 scr2()

```
void scr2 (
            double vk,
            double vkarg,
            double exri,
            double * duk,
            ParticleDescriptor * c1 )
```

This function computes the scattering amplitude for single spheres constituting an aggregate. See Sec. 4.2.1 in Borghese, Denti & Saija (2007).

Parameters

vk	`double` Wave number.
vkarg	`double` ANNOTATION: Argument of wave number.
exri	`double` External medium refractive index.
duk	`double *` ANNOTATION: variation of unit wave vector.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.5.2.23 str()

```
void str (
            double ** rcf,
            ParticleDescriptor * c1 )
```

This function transforms the Cartesian coordinates of the spheres in an aggregate to radial coordinates, then it calls `sphar()` to calculate the vector of spherical harmonics of the incident field.

Parameters

rcf	`double **` Matrix of sphere configuration fractional radii.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.5.2.24 tqr()

```
void tqr (
            double * u,
            double * up,
            double * un,
            double * tqev,
            double * tqsv,
            double & tep,
            double & ten,
            double & tek,
            double & tsp,
            double & tsn,
            double & tsk )
```

This function computes the radiation torques resulting from the difference between absorption and scattering contributions, like `rabas()`, but for a coordinate system oriented along the wave vector and its orthogonal axis. See Sec. 4.9 in Borghese, Denti & Saija (2007).

Parameters

u	`double *`
up	`double *`
un	`double *`
tqev	`double *`
tqsv	`double *`
tep	`double &`
ten	`double &`
tek	`double &`
tsp	`double &`
tsn	`double &`
tsk	`double &`

9.5.2.25 ztm()

```
void ztm (
            dcomplex ** am,
            ParticleDescriptor * c1 )
```

This function computes the single-centered inverted M-matrix appearing in Eq. (5.28) of Borghese, Denti & Saija (2007).

Parameters

am	`complex double **`
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.6 clu_subs.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00031 #ifndef INCLUDE_CLU_SUBS_H_
00032 #define INCLUDE_CLU_SUBS_H_
00033
00048 void apc(
00049      double ****zpv, int le, dcomplex **am0m, dcomplex **w,
00050      double sqk, double **gapr, dcomplex **gapp
00051 );
00052
00067 void apcra(
00068        double ****zpv, const int le, dcomplex **am0m, int inpol, double sqk,
00069        double **gaprm, dcomplex **gappm
00070 );
00071
00084 dcomplex cdtp(dcomplex z, dcomplex *vec_am, int i, int jf, int k, int nj, np_int istep);
00085
00094 double cgev(int ipamo, int mu, int l, int m);
00095
00104 void cms(dcomplex **am, ParticleDescriptor *c1);
00105
00116 void crsm1(double vk, double exri, ParticleDescriptor *c1);
00117
00133 dcomplex ghit_d(
00134           int ihi, int ipamo, int nbl, int l1, int m1, int l2, int m2,
00135           ParticleDescriptor *c1, double *rac3j
00136 );
00137
00152 dcomplex ghit(
00153           int ihi, int ipamo, int nbl, int l1, int m1, int l2, int m2,
00154           ParticleDescriptor *c1
00155 );
00156
00169 void hjv(
00170      double exri, double vk, int &jer, int &lcalc, dcomplex &arg,
00171      ParticleDescriptor *c1
00172 );
00173
00184 void lucin(dcomplex **am, const np_int nddmst, np_int n, int &ier);
00185
00198 void mextc(double vk, double exri, dcomplex **fsac, double **cextlr, double **cext);
00199
00210 void pcros(double vk, double exri, ParticleDescriptor *c1);
00211
00222 void pcrsm0(double vk, double exri, int inpol, ParticleDescriptor *c1);
00223
00238 void polar(
00239        double x, double y, double z, double &r, double &cth, double &sth,
00240        double &cph, double &sph
00241 );
00242
00252 void r3j000(int j2, int j3, double *rac3j);
00253
00265 void r3jjr(int j2, int j3, int m2, int m3, double *rac3j);
00266
00278 void r3jjr_d(int j2, int j3, int m2, int m3, double *rac3j);
00279
00291 void r3jmr(int j1, int j2, int j3, int m1, double *rac3j);
00292
00308 void raba(
00309       int le, dcomplex **am0m, dcomplex **w, double **tqce,
00310       dcomplex **tqcpe, double **tqcs, dcomplex **tqcps
00311 );
00312
00333 void rftr(
00334       double *u, double *up, double *un, double *gapv, double extins, double scatts,
00335       double &rapr, double &cosav, double &fp, double &fn, double &fk, double &fx,
00336       double &fy, double &fz
00337 );
00338
00350 void scr0(double vk, double exri, ParticleDescriptor *c1);
00351
00363 void scr2(
```

```
00364       double vk, double vkarg, double exri, double *duk, ParticleDescriptor *c1
00365 );
00366
00376 void str(double **rcf, ParticleDescriptor *c1);
00377
00397 void tqr(
00398      double *u, double *up, double *un, double *tqev, double *tqsv, double &tep,
00399      double &ten, double &tek, double &tsp, double &tsn, double &tsk
00400 );
00401
00410 void ztm(dcomplex **am, ParticleDescriptor *c1);
00411
00412 #endif
```

9.7 np_tmcode/src/include/Commons.h File Reference

C++ porting of common data structures.

Classes

- class mixMPI

  Structure with essential MPI data.
- class ParticleDescriptor

  Basic data structure describing the particle model and its interaction with fields.
- class ParticleDescriptorCluster

  The data structure describing a particle model made by a cluster of spheres.
- class ParticleDescriptorInclusion

  The data structure describing a particle model for a sphere with inclusions.
- class ParticleDescriptorSphere

  The data structure describing a spherical particle model.
- class ScatteringAngles

  A data structure representing the angles to be evaluated in the problem.

9.7.1 Detailed Description

Many functions of the original FORTRAN code share complex data blocks in form of COMMON blocks. This poses the limit of freezing the structure of the data blocks in the code, therefore implying the necessity to modify the code to adapt it to the input and to recompile before running. C++, on the contrary, offers the possibility to represent the necessary data structures as classes that can dynamically instantiate the shared information in the most convenient format for the current configuration. This approach adds an abstraction layer that lifts the need to modify and recompile the code depending on the input.

9.8 Commons.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
```

```
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00033 #ifndef INCLUDE_COMMONS_H_
00034 #define INCLUDE_COMMONS_H_
00035
00036 class ParticleDescriptor;
00037
00040 class mixMPI {
00041 public:
00043   bool mpirunning;
00045   int rank;
00047   int nprocs;
00048
00051   mixMPI();
00052
00055 #ifdef MPI_VERSION
00056   mixMPI(MPI_Comm comm);
00057 #endif
00058
00061   mixMPI(const mixMPI& rhs);
00062
00065   ~mixMPI();
00066 };
00067
00074 class ParticleDescriptor {
00075 protected:
00076   // »> COMMON TO ALL DESCRIPTOR TYPES «< //
00078   short _class_type;
00080   int _nsph;
00082   int _li;
00084   int _max_layers;
00086   int _num_configurations;
00088   int _num_layers;
00090   int _nhspo;
00092   int _npnt;
00094   int _npntts;
00096   dcomplex *vec_rmi;
00098   dcomplex *vec_rei;
00100   dcomplex *vec_w;
00102   double *vec_rc;
00103   // »> END OF SECTION COMMON TO ALL DESCRIPTOR TYPES «< //
00104
00105   // »> NEEDED BY SPHERE AND CLUSTER «< //
00107   dcomplex *vec_sas;
00109   dcomplex *vec_vints;
00110   // »> END OF SECTION NEEDED BY SPHERE AND CLUSTER «< //
00111
00112   // »> NEEDED BY CLUSTER «< //
00114   dcomplex *vec_tsas;
00116   dcomplex *vec_gis;
00118   dcomplex *vec_gls;
00120   dcomplex *vec_sam;
00121   // »> END OF SECTION NEEDED BY CLUSTER «< //
00122
00123   // »> NEEDED BY CLUSTER AND INCLU «<
00125   int _le;
00127   int _lm;
00129   int _nlim;
00131   int _nlem;
00133   int _nlemt;
00135   int _ncou;
00137   int _litpo;
00139   int _litpos;
00141   int _lmpo;
00143   int _lmtpo;
00145   int _lmtpos;
00147   int _nv3j;
00149   int _ndi;
00151   int _ndit;
00153   dcomplex *vec_am0m;
00155   dcomplex *vec_fsac;
00157   dcomplex *vec_sac;
00159   dcomplex *vec_fsacm;
00161   int *vec_ind3j;
00162   // »> END OF SECTION NEEDED BY CLUSTER AND INCLU «< //
00163
00164   // »> NEEDED BY INCLU «< //
00166   int _ndm;
00168   dcomplex *vec_at;
00169   // »> END OF SECTION NEEDED BY INCLU «< //
00170 public:
00171   // »> COMMON TO ALL DESCRIPTOR TYPES «< //
00173   const static short BASE_TYPE = 0;
00175   const static short SPHERE_TYPE = 1;
```

```
00177   const static short CLUSTER_TYPE = 2;
00179   const static short INCLUSION_TYPE = 3;
00180
00182   const short& class_type = _class_type;
00184   const int &nsph = _nsph;
00186   const int &li = _li;
00188   const int &max_layers = _max_layers;
00190   const int &num_configurations = _num_configurations;
00192   const int &num_layers = _num_layers;
00194   const int &nhspo = _nhspo;
00196   const int &npnt = _npnt;
00198   const int &npntts = _npntts;
00200   dcomplex **rmi;
00202   dcomplex **rei;
00204   dcomplex **w;
00206   dcomplex *vint;
00208   double *rxx;
00210   double *ryy;
00212   double *rzz;
00214   double *ros;
00216   double **rc;
00218   int *iog;
00220   int *nshl;
00222   dcomplex *ris;
00224   dcomplex *dlri;
00226   dcomplex *dc0;
00228   dcomplex *vkt;
00230   double *vsz;
00232   double gcs;
00233   // »> END OF SECTION COMMON TO ALL DESCRIPTOR TYPES «< //
00234
00235   // »> NEEDED BY SPHERE AND CLUSTER «< //
00237   dcomplex ***sas;
00239   dcomplex **vints;
00241   dcomplex *fsas;
00243   double *sscs;
00245   double *sexs;
00247   double *sabs;
00249   double *sqscs;
00251   double *sqexs;
00253   double *sqabs;
00255   double *gcsv;
00256   // »> END OF SECTION NEEDED BY SPHERE AND CLUSTER «< //
00257
00258   // »> NEEDED BY CLUSTER «<
00260   dcomplex *vintt;
00262   dcomplex tfsas;
00264   dcomplex **tsas;
00266   double scs;
00268   double ecs;
00270   double acs;
00272   dcomplex **gis;
00274   dcomplex **gls;
00276   dcomplex **sam;
00277   // »> END OF SECTION NEEDED BY CLUSTER «< //
00278
00279   // »> NEEDED BY CLUSTER AND INCLU «<
00281   const int& le = _le;
00283   const int& lm = _lm;
00285   const int& nlim = _nlim;
00287   const int& nlem = _nlem;
00289   const int& nlemt = _nlemt;
00291   const int& ncou = _ncou;
00293   const int& litpo = _litpo;
00295   const int& litpos = _litpos;
00297   const int& lmpo = _lmpo;
00299   const int& lmtpo = _lmtpo;
00301   const int& lmtpos = _lmtpos;
00303   const int& nv3j = _nv3j;
00305   const int& ndi = _ndi;
00307   const int& ndit = _ndit;
00309   const int& ndm = _ndm;
00310
00312   dcomplex *vh;
00314   dcomplex *vj0;
00316   dcomplex *vyhj;
00318   dcomplex *vyj0;
00320   dcomplex vj;
00322   dcomplex **am0m;
00324   dcomplex **fsac;
00326   dcomplex **sac;
00328   dcomplex **fsacm;
00330   dcomplex *vintm;
00332   dcomplex *scscp;
00334   dcomplex *ecscp;
00336   dcomplex *scscpm;
00338   dcomplex *ecscpm;
```

```
00340   double *v3j0;
00342   double *scsc;
00344   double *ecsc;
00346   double *scscm;
00348   double *ecscm;
00350   int **ind3j;
00352   double *rac3j;
00353   // »> END OF SECTION NEEDED BY CLUSTER AND INCLU «< //
00354
00355   // »> NEEDED BY INCLU «< //
00357   dcomplex *rm0;
00359   dcomplex *re0;
00361   dcomplex *rmw;
00363   dcomplex *rew;
00365   dcomplex *tm;
00367   dcomplex *te;
00369   dcomplex *tm0;
00371   dcomplex *te0;
00373   dcomplex **at;
00374   // »> END OF SECTION NEEDED BY INCLU «< //
00375
00381   ParticleDescriptor(GeometryConfiguration *gconf, ScattererConfiguration *sconf);
00382
00387   ParticleDescriptor(const ParticleDescriptor &rhs);
00388
00389 #ifdef MPI_VERSION
00394   ParticleDescriptor(const mixMPI *mpidata);
00395
00400   void mpibcast(const mixMPI *mpidata);
00401 #endif // MPI_VERSION
00402
00405   ~ParticleDescriptor();
00406
00411   virtual std::string get_descriptor_type() { return "base descriptor"; }
00412
00419   static long get_size(GeometryConfiguration *gconf, ScattererConfiguration *sconf);
00420 };
00421
00428 class ParticleDescriptorCluster: public ParticleDescriptor {
00429 public:
00435   ParticleDescriptorCluster(GeometryConfiguration *gconf, ScattererConfiguration *sconf);
00436
00441   ParticleDescriptorCluster(const ParticleDescriptorCluster &rhs);
00442
00443 #ifdef MPI_VERSION
00448   ParticleDescriptorCluster(const mixMPI *mpidata);
00449
00454   void mpibcast(const mixMPI *mpidata);
00455 #endif // MPI_VERSION
00456
00461   std::string get_descriptor_type() override { return "cluster descriptor"; }
00462
00469   static long get_size(GeometryConfiguration *gconf, ScattererConfiguration *sconf);
00470
00477   int update_orders(int inner_order, int outer_order);
00478 };
00479
00486 class ParticleDescriptorInclusion: public ParticleDescriptor {
00487 public:
00493   ParticleDescriptorInclusion(GeometryConfiguration *gconf, ScattererConfiguration *sconf);
00494
00499   ParticleDescriptorInclusion(const ParticleDescriptorInclusion &rhs);
00500
00501 #ifdef MPI_VERSION
00506   ParticleDescriptorInclusion(const mixMPI *mpidata);
00507
00512   void mpibcast(const mixMPI *mpidata);
00513 #endif // MPI_VERSION
00514
00519   std::string get_descriptor_type() override { return "inclusion descriptor"; }
00520
00527   static long get_size(GeometryConfiguration *gconf, ScattererConfiguration *sconf);
00528
00535   int update_orders(int inner_order, int outer_order);
00536 };
00537
00544 class ParticleDescriptorSphere: public ParticleDescriptor {
00545 public:
00551   ParticleDescriptorSphere(GeometryConfiguration *gconf, ScattererConfiguration *sconf);
00552
00557   ParticleDescriptorSphere(const ParticleDescriptorSphere &rhs);
00558
00559 #ifdef MPI_VERSION
00564   ParticleDescriptorSphere(const mixMPI *mpidata);
00565
00570   void mpibcast(const mixMPI *mpidata);
00571 #endif // MPI_VERSION
```

```
00572
00577   std::string get_descriptor_type() override { return "sphere descriptor"; }
00578
00585   static long get_size(GeometryConfiguration *gconf, ScattererConfiguration *sconf);
00586
00592   int update_order(int order);
00593 };
00594
00598 class ScatteringAngles {
00599 protected:
00601   int _nth;
00603   int _nths;
00605   int _nph;
00607   int _nphs;
00609   int _nk;
00611   int _nks;
00613   int _nkks;
00615   double _th;
00617   double _thstp;
00619   double _thlst;
00621   double _ths;
00623   double _thsstp;
00625   double _thslst;
00627   double _ph;
00629   double _phstp;
00631   double _phlst;
00633   double _phs;
00635   double _phsstp;
00637   double _phslst;
00639   double _thsca;
00640
00641 public:
00643   const int& nth = _nth;
00645   const int& nths = _nths;
00647   const int& nph = _nph;
00649   const int& nphs = _nphs;
00651   const int& nk = _nk;
00653   const int& nks = _nks;
00655   const int& nkks = _nkks;
00657   const double& th = _th;
00659   const double& thstp = _thstp;
00661   const double& thlst = _thlst;
00663   const double& ths = _ths;
00665   const double& thsstp = _thsstp;
00667   const double& thslst = _thslst;
00669   const double& ph = _ph;
00671   const double& phstp = _phstp;
00673   const double& phlst = _phlst;
00675   const double& phs = _phs;
00677   const double& phsstp = _phsstp;
00679   const double& phslst = _phslst;
00681   const double& thsca = _thsca;
00682
00687   ScatteringAngles(GeometryConfiguration *gconf);
00688
00693   ScatteringAngles(const ScatteringAngles &rhs);
00694
00695 #ifdef MPI_VERSION
00700   ScatteringAngles(const mixMPI *mpidata);
00701
00711   void mpibcast(const mixMPI *mpidata);
00712 #endif
00713
00714 };
00715
00716
00717 #endif
```

9.9 np_tmcode/src/include/Configuration.h File Reference

Configuration data structures.

Classes

- class GeometryConfiguration

  A class to represent the configuration of the scattering geometry.
- class ScattererConfiguration

  A class to represent scatterer configuration objects.

Functions

- int * check_overlaps (ScattererConfiguration *sconf, GeometryConfiguration *gconf, const double tolerance=0.0)

  Check the presence of overlapping spheres in an aggregate.
- double get_overlap (ScattererConfiguration *sconf, GeometryConfiguration *gconf, const int index_0, const int index_1)

  Get the overlap among two spheres.

9.9.1 Detailed Description

To handle the calculation of a scattering problem, the code needs a set of configuration parameters that define the properties of the scattering particle, of the incident radiation field and of the geometry of the problem. The necessary information is managed through the use of two classes: `ScattererConfiguration` and `GeometryConfiguration`. The first class, `ScattererConfiguration`, undertakes the role of describing the scattering particle properties, by defining its structure in terms of a distribution of spherical sub-particles, and the optical properties of the constituting materials. It also defines the scaling vector, which tracks the ratio of particle size and radiation wavelength, eventually allowing for the iteration of the calculation on different wavelengths. Multiple materials and layered structures are allowed, while sphere compenetration is not handled, due to the implied discontinuity on the spherical symmetry of the elementary sub-particles. The second class, `GeometryConfiguration`, on the contrary, describes the properties of the radiation field impinging on the particle. It defines the incoming radiation polarization state, the incident direction and the scattering directions that need to be accounted for. Both classes expose methods to read the required configuration data from input files formatted according to the same rules expected by the original code. In addition, they offer perform I/O operations towards proprietary and standard binary formats.

9.9.2 Function Documentation

9.9.2.1 check_overlaps()

```
int * check_overlaps (
            ScattererConfiguration * sconf,
            GeometryConfiguration * gconf,
            const double tolerance = 0.0 )
```

The theoretical implementation of the code is based on the assumption that the spherical monomers that compose the aggregate do not overlap in space. Small violations of this assumption do not critically affect the results, but they need to be reported. This function performs a scan of the aggregate and returns a summary of any potential overlap, if detected.

Parameters

sconf	`ScattererConfiguration *` Pointer to a ScattererConfiguration instance.
gconf	`GeometryConfiguration *` Pointer to a GeometryConfiguration instance.
tolerance	`const double` A tolerance value below which overlap is ignored.

Returns

overlaps: `int *` A vector containing the number of overlaps and a list of overlapping sphere indices.

9.9.2.2 get_overlap()

```
double get_overlap (
            ScattererConfiguration * sconf,
            GeometryConfiguration * gconf,
            const int index_0,
            const int index_1 )
```

This function computes the thickness of the compenetration between two spheres, in order to compare it with the current tolerance settings.

Parameters

sconf	`ScattererConfiguration *` Pointer to a ScattererConfiguration instance.
gconf	`GeometryConfiguration *` Pointer to a GeometryConfiguration instance.
index_0	`const int` Index of the first sphere.
index_1	`const int` Index of the second sphere.

Returns

overlap: `double` The thickness of the overlapping layer in meters.

9.10 Configuration.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00044 #ifndef INCLUDE_CONFIGURATION_H_
00045 #define INCLUDE_CONFIGURATION_H_
00046
00047 class mixMPI;
00048
00057 class GeometryConfiguration {
00058
00059 protected:
00061   int _number_of_spheres;
00063   int _l_max;
00065   int _li;
00067   int _le;
00069   np_int _mxndm;
00071   int _iavm;
00073   int _in_pol;
00075   int _npnt;
00077   int _npntts;
00079   int _isam;
00081   int _jwtm;
00083   double _in_theta_start;
00085   double _in_theta_step;
00087   double _in_theta_end;
00089   double _sc_theta_start;
```

```
00091   double _sc_theta_step;
00093   double _sc_theta_end;
00095   double _in_phi_start;
00097   double _in_phi_step;
00099   double _in_phi_end;
00101   double _sc_phi_start;
00103   double _sc_phi_step;
00105   double _sc_phi_end;
00107   double *_sph_x;
00109   double *_sph_y;
00111   double *_sph_z;
00113   short _refine_flag;
00115   short _dyn_order_flag;
00117   double _host_ram_gb;
00119   double _gpu_ram_gb;
00121   double _tolerance;
00122
00123 public:
00125   const int& number_of_spheres = _number_of_spheres;
00127   const int& l_max = _l_max;
00129   const int& li = _li;
00131   const int& le = _le;
00133   const np_int& mxndm = _mxndm;
00135   const int& iavm = _iavm;
00137   const int& in_pol = _in_pol;
00139   const int& npnt = _npnt;
00141   const int& npntts = _npntts;
00143   const int& isam = _isam;
00145   const int& jwtm = _jwtm;
00147   const double& in_theta_start = _in_theta_start;
00149   const double& in_theta_step = _in_theta_step;
00151   const double& in_theta_end = _in_theta_end;
00153   const double& sc_theta_start = _sc_theta_start;
00155   const double& sc_theta_step = _sc_theta_step;
00157   const double& sc_theta_end = _sc_theta_end;
00159   const double& in_phi_start = _in_phi_start;
00161   const double& in_phi_step = _in_phi_step;
00163   const double& in_phi_end = _in_phi_end;
00165   const double& sc_phi_start = _sc_phi_start;
00167   const double& sc_phi_step = _sc_phi_step;
00169   const double& sc_phi_end = _sc_phi_end;
00171   const short& refine_flag = _refine_flag;
00173   const short& dyn_order_flag = _dyn_order_flag;
00175   const double& host_ram_gb = _host_ram_gb;
00177   const double& gpu_ram_gb = _gpu_ram_gb;
00179   const double& tolerance = _tolerance;
00180
00213   GeometryConfiguration(
00214             int nsph, int lm, int in_pol, int npnt, int npntts, int meridional_type,
00215             int li, int le, np_int mxndm, int iavm,
00216             double *x, double *y, double *z,
00217             double in_th_start, double in_th_step, double in_th_end,
00218             double sc_th_start, double sc_th_step, double sc_th_end,
00219             double in_ph_start, double in_ph_step, double in_ph_end,
00220             double sc_ph_start, double sc_ph_step, double sc_ph_end,
00221             int jwtm
00222             );
00223
00228   GeometryConfiguration(const GeometryConfiguration& rhs);
00229
00230 #ifdef MPI_VERSION
00235   GeometryConfiguration(const mixMPI *mpidata);
00236
00246   void mpibcast(const mixMPI *mpidata);
00247 #endif
00248
00251   ~GeometryConfiguration();
00252
00264   static GeometryConfiguration *from_legacy(const std::string& file_name);
00265
00274   double get_sph_x(int index) { return _sph_x[index]; }
00275
00284   double get_sph_y(int index) { return _sph_y[index]; }
00285
00294   double get_sph_z(int index) { return _sph_z[index]; }
00295
00296 };
00297
00304 class ScattererConfiguration {
00305
00306 protected:
00308   dcomplex ***_dc0_matrix;
00310   double *_radii_of_spheres;
00312   int *_iog_vec;
00314   int *_nshl_vec;
00316   double *_scale_vec;
00318   std::string _reference_variable_name;
```

```
00320   int _number_of_spheres;
00322   int _configurations;
00324   int _number_of_scales;
00326   int _idfc;
00328   double _exdc;
00330   double _wp;
00332   double _xip;
00334   int _max_layers;
00336   bool _use_external_sphere;
00337
00348   static ScattererConfiguration *from_hdf5(const std::string& file_name);
00349
00361   static ScattererConfiguration *from_legacy(const std::string& file_name);
00362
00371   void write_hdf5(const std::string& file_name);
00372
00382   void write_legacy(const std::string& file_name);
00383 public:
00385   const std::string& reference_variable_name = _reference_variable_name;
00387   const int& number_of_spheres = _number_of_spheres;
00389   const int& configurations = _configurations;
00391   const int& number_of_scales = _number_of_scales;
00393   const int& idfc = _idfc;
00395   const double& exdc = _exdc;
00397   const double& wp = _wp;
00399   const double& xip = _xip;
00401   const int& max_layers = _max_layers;
00403   const bool& use_external_sphere = _use_external_sphere;
00405   double **_rcf;
00406
00430   ScattererConfiguration(
00431             int nsph,
00432             int configs,
00433             double *scale_vector,
00434             int nxi,
00435             const std::string& variable_name,
00436             int *iog_vector,
00437             double *ros_vector,
00438             int *nshl_vector,
00439             double **rcf_vector,
00440             int dielectric_func_type,
00441             dcomplex ***dc_matrix,
00442             bool has_external,
00443             double exdc,
00444             double wp,
00445             double xip
00446   );
00447
00456   ScattererConfiguration(const ScattererConfiguration& rhs);
00457
00458 #ifdef MPI_VERSION
00463   ScattererConfiguration(const mixMPI *mpidata);
00464
00474   void mpibcast(const mixMPI *mpidata);
00475 #endif
00476
00479   ~ScattererConfiguration();
00480
00496   static ScattererConfiguration* from_binary(const std::string& file_name, const std::string& mode =
      "LEGACY");
00497
00509   static ScattererConfiguration* from_dedfb(const std::string& file_name);
00510
00522   dcomplex get_dielectric_constant(int i, int j, int k) { return _dc0_matrix[i][j][k]; }
00523
00532   int get_iog(int index) { return _iog_vec[index]; }
00533
00538   double get_max_radius();
00539
00544   double get_min_radius();
00545
00554   int get_nshl(int index) { return _nshl_vec[index]; }
00555
00561   double get_particle_radius(GeometryConfiguration *gc);
00562
00571   double get_radius(int index);
00572
00582   double get_rcf(int row, int column) { return _rcf[row][column]; }
00583
00592   double get_scale(int index) { return _scale_vec[index]; }
00593
00602   double get_type_radius(int index) { return _radii_of_spheres[index]; }
00603
00609   void print();
00610
00624   void write_binary(const std::string& file_name, const std::string& mode="LEGACY");
00625
```

```
00636   void write_formatted(const std::string& file_name);
00637
00649   bool operator ==(const ScattererConfiguration &other);
00650
00651 };
00652
00667 int *check_overlaps(
00668   ScattererConfiguration *sconf, GeometryConfiguration *gconf, const double tolerance=0.0
00669 );
00670
00682 double get_overlap(
00683   ScattererConfiguration *sconf, GeometryConfiguration *gconf,
00684   const int index_0, const int index_1
00685 );
00686
00687 #endif // INCLUDE_CONFIGURATION_H_
```

9.11 np_tmcode/src/include/cublas_calls.h File Reference

C++ interface to CUBLAS calls.

Functions

- void cublas_zinvert (dcomplex **mat, np_int n, int device_id)

  Invert a complex matrix with double precision elements.
- void cublas_zinvert_and_refine (dcomplex **mat, np_int n, int &maxrefiters, double &accuracygoal, int refinemode, int device_id)

  Invert a complex matrix with double precision elements, applying iterative refinement of the solution.

9.11.1 Function Documentation

9.11.1.1 cublas_zinvert()

```
void cublas_zinvert (
            dcomplex ** mat,
            np_int n,
            int device_id )
```

Use CUBLAS to perform an in-place matrix inversion for a complex matrix with double precision elements.

Parameters

mat	Matrix of complex. The matrix to be inverted.
n	np_int The number of rows and columns of the [n x n] matrix.
device_id	int ID of the device for matrix inversion offloading.

9.11.1.2 cublas_zinvert_and_refine()

```
void cublas_zinvert_and_refine (
            dcomplex ** mat,
            np_int n,
```

```
            int & maxrefiters,
            double & accuracygoal,
            int refinemode,
            int device_id )
```

Use CUBLAS to perform matrix inversion for a complex matrix with double precision elements.

Parameters

mat	Matrix of complex. The matrix to be inverted.
n	`np_int` The number of rows and columns of the [n x n] matrix.
maxrefiters	`int` Maximum number of refinement iterations to apply.
accuracygoal	`double &` Accuracy to achieve in iterative refinement, defined as the module of the maximum difference between the identity matrix and the matrix product of the (approximate) inverse times the original matrix. On return, it contains the actually achieved accuracy.
refinemode	`int` Flag to choose the refinement mode.
device_id	`int` ID of the device for matrix inversion offloading.

9.12 cublas_calls.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00023 #ifndef INCLUDE_CUBLAS_CALLS_H_
00024 #define INCLUDE_CUBLAS_CALLS_H_
00025
00035 void cublas_zinvert(dcomplex **mat, np_int n, int device_id);
00036
00049 void cublas_zinvert_and_refine(dcomplex **mat, np_int n, int &maxrefiters, double &accuracygoal, int
      refinemode, int device_id);
00050
00051 #endif
```

9.13 np_tmcode/src/include/errors.h File Reference

Collection of proprietary code exceptions.

Classes

- class UnrecognizedOutputInfo

  Exception for wrong OutputInfo NULL calls.
- class ListOutOfBoundsException

  Exception for out of bounds List requests.

- class ObjectAllocationException

  Exception for object allocation error handlers.
- class OpenConfigurationFileException

  Exception for open file error handlers.
- class MatrixOutOfBoundsException

  Exception for access requests out of matrix bounds.
- class UnrecognizedConfigurationException

  Exception for unrecognized configuration data sets.
- class UnrecognizedFormatException

  Exception for unrecognized file formats.
- class UnrecognizedParameterException

  Exception for unrecognized parameters.

9.13.1 Detailed Description

There are many circumstances that can prevent the correct execution of a code. These range from user mistakes, to improper configuration, to unsupported hardware and all the way up to various system failures. Although it is not possible to grant proper execution in all cases, it is often possible to design a code in such a way that the program detects unexpected conditions, informs the user and takes the proper actions, eventually stopping without crash, if no other options are available. C++ handles such unexpected circumstances by means of `exceptions`. These are special procedures that can be launched whenever an unexpected situation occurs and they allow to restore the code work-flow and attempt recovery. Exceptions can be divided in different cathegories, which respond to various types of problems. This library contains a set of exceptions designed to the most common problems that may occur while executing an application of the `NP_TMcode` suite.

9.14 errors.h

Go to the documentation of this file.

```
00001
00022 #ifndef INCLUDE_ERRORS_H_
00023 #define INCLUDE_ERRORS_H_
00024
00027 class UnrecognizedOutputInfo: public std::exception {
00028 protected:
00030   std::string message;
00031
00032 public:
00038   UnrecognizedOutputInfo(int requested) {
00039     message = "Error: passed parameter " + std::to_string(requested)
00040       + ", but only 1 is allowed";
00041   }
00042
00046   virtual const char* what() const throw() {
00047     return message.c_str();
00048   }
00049 };
00050
00053 class ListOutOfBoundsException: public std::exception {
00054 protected:
00056   std::string message;
00057
00058 public:
00066   ListOutOfBoundsException(int requested, int min, int max) {
00067     message = "Error: requested index " + std::to_string(requested)
00068       + " out of list allowed bounds [" + std::to_string(min) + ", "
00069       + std::to_string(max - 1) + "]";
00070   }
00071
00075   virtual const char* what() const throw() {
00076     return message.c_str();
00077   }
00078 };
00079
00082 class ObjectAllocationException: public std::exception {
```

```
00083 protected:
00085   std::string file_name;
00086
00087 public:
00093   ObjectAllocationException(const std::string& name) { file_name = name; }
00094
00098   virtual const char* what() const throw() {
00099     return file_name.c_str();
00100   }
00101 };
00102
00105 class OpenConfigurationFileException: public std::exception {
00106 protected:
00108   std::string file_name;
00109
00110 public:
00116   OpenConfigurationFileException(const std::string& name) { file_name = name; }
00117
00121   virtual const char* what() const throw() {
00122     return file_name.c_str();
00123   }
00124 };
00125
00128 class MatrixOutOfBoundsException: public std::exception {
00129 protected:
00131   std::string message;
00132 public:
00138   MatrixOutOfBoundsException(const std::string& problem) { message = problem; }
00142   virtual const char* what() const throw() {
00143     return message.c_str();
00144   }
00145 };
00146
00149 class UnrecognizedConfigurationException: public std::exception {
00150 protected:
00152   std::string message;
00153 public:
00159   UnrecognizedConfigurationException(const std::string& problem) { message = problem; }
00163   virtual const char* what() const throw() {
00164     return message.c_str();
00165   }
00166 };
00167
00170 class UnrecognizedFormatException: public std::exception {
00171 protected:
00173   std::string message;
00174 public:
00180   UnrecognizedFormatException(const std::string& problem) { message = problem; }
00184   virtual const char* what() const throw() {
00185     return message.c_str();
00186   }
00187 };
00188
00191 class UnrecognizedParameterException: public std::exception {
00192 protected:
00194   std::string message;
00195 public:
00201   UnrecognizedParameterException(const std::string& problem) { message = problem; }
00205   virtual const char* what() const throw() {
00206     return message.c_str();
00207   }
00208 };
00209
00210 #endif
```

9.15 np_tmcode/src/include/file_io.h File Reference

Library to handle I/O operations with files.

```
#include <vector>
```

Classes

- class FileSchema

File content descriptor.

- class HDFFile

  HDF5 I/O wrapper class.

- class VirtualAsciiFile

  Virtual representation of an ASCII file.

- class VirtualBinaryLine

  Virtual representation of a binary file line.

- class VirtualBinaryFile

  Virtual representation of a binary file.

9.16 file_io.h

Go to the documentation of this file.

```
00001
00005 #ifndef INCLUDE_FILE_IO_H_
00006 #define INCLUDE_FILE_IO_H_
00007
00008 #include <vector>
00009
00010 class mixMPI;
00011
00022 class FileSchema {
00023  protected:
00025   int num_records;
00027   std::string *record_names;
00029   std::string *record_types;
00030
00031  public:
00038   FileSchema(int num_rec, const std::string *rec_types, const std::string *rec_names=NULL);
00039
00042   ~FileSchema();
00043
00048   int get_record_number() { return num_records; }
00049
00054   std::string *get_record_names();
00055
00060   std::string *get_record_types();
00061 };
00062
00069 class HDFFile {
00070  protected:
00072   List<hid_t> *id_list;
00074   std::string file_name;
00076   bool file_open_flag;
00078   hid_t file_id;
00080   herr_t status;
00081
00082  public:
00090   HDFFile(
00091       const std::string& name, unsigned int flags=H5F_ACC_EXCL,
00092       hid_t fcpl_id=H5P_DEFAULT, hid_t fapl_id=H5P_DEFAULT
00093   );
00094
00097   ~HDFFile();
00098
00101   herr_t close();
00102
00112   static HDFFile* from_schema(
00113                   FileSchema &schema, const std::string& name, unsigned int flags=H5F_ACC_EXCL,
00114                   hid_t fcpl_id=H5P_DEFAULT, hid_t fapl_id=H5P_DEFAULT
00115   );
00116
00119   hid_t get_file_id() { return file_id; }
00120
00123   herr_t get_status() { return status; }
00124
00127   bool is_open() { return file_open_flag; }
00128
00140   herr_t read(
00141            const std::string& dataset_name, const std::string& data_type, void *buffer,
00142            hid_t mem_space_id=H5S_ALL, hid_t file_space_id=H5S_ALL,
00143            hid_t dapl_id=H5P_DEFAULT, hid_t dxpl_id=H5P_DEFAULT
00144   );
00145
00157   herr_t write(
```

```
00158             const std::string& dataset_name, const std::string& data_type, const void *buffer,
00159             hid_t mem_space_id=H5S_ALL, hid_t file_space_id=H5S_ALL,
00160             hid_t dapl_id=H5P_DEFAULT, hid_t dxpl_id=H5P_DEFAULT
00161   );
00162 };
00163
00168 class VirtualAsciiFile {
00169 protected:
00171   // int32_t _num_lines;
00173   std::vector<std::string> *_file_lines;
00174
00175 public:
00176   // const int32_t &num_lines = _num_lines;
00181   VirtualAsciiFile(int32_t lines = 0);
00182
00187   VirtualAsciiFile(const VirtualAsciiFile& rhs);
00188
00194   VirtualAsciiFile(const mixMPI *mpidata, int rr);
00195
00198   ~VirtualAsciiFile();
00199
00204   void append(VirtualAsciiFile& rhs);
00205
00210   void append_line(const std::string& line);
00211
00217   int append_to_disk(const std::string& file_name);
00218
00233   int insert(int32_t position, VirtualAsciiFile& rhs, int32_t start = 0, int32_t end = 0);
00234
00239   int32_t number_of_lines() { return _file_lines->size(); }
00240
00246   int write_to_disk(const std::string& file_name);
00247
00252   void mpisend(const mixMPI *mpidata);
00253 };
00254
00255
00260 class VirtualBinaryLine {
00261 // protected:
00262 //   //! \brief The pointer to the piece of data to be written, cast to char *
00263 //   char *_data_pointer;
00264 //   //! \brief the size of the data block.
00265 //   size_t _data_size;
00266
00267 public:
00269   char *_data_pointer;
00271   size_t _data_size;
00273   const char* data_pointer = _data_pointer;
00275   const size_t & data_size = _data_size;
00276
00281   VirtualBinaryLine(int mydata);
00282
00287   VirtualBinaryLine(long mydata);
00288
00293   VirtualBinaryLine(float mydata);
00294
00299   VirtualBinaryLine(double mydata);
00300
00305   VirtualBinaryLine(dcomplex mydata);
00306
00311   VirtualBinaryLine(const VirtualBinaryLine& rhs);
00312
00318   VirtualBinaryLine(const mixMPI *mpidata, int rr);
00319
00322   ~VirtualBinaryLine();
00323
00328   void mpisend(const mixMPI *mpidata);
00329 };
00330
00331
00336 class VirtualBinaryFile {
00337 protected:
00339   // int32_t _num_lines;
00340   // //! \brief A vector of strings representing the file lines.
00341   // std::vector<VirtualBinaryLine> *_file_lines;
00342
00343 public:
00345   std::vector<VirtualBinaryLine> *_file_lines;
00346   // const int32_t &num_lines = _num_lines;
00350   VirtualBinaryFile();
00351
00356   VirtualBinaryFile(const VirtualBinaryFile& rhs);
00357
00363   VirtualBinaryFile(const mixMPI *mpidata, int rr);
00364
00367   ~VirtualBinaryFile();
00368
```

```
00373   void append(VirtualBinaryFile& rhs);
00374
00379   void append_line(const VirtualBinaryLine& line);
00380
00386   int append_to_disk(const std::string& file_name);
00387
00392   int32_t number_of_lines() { return _file_lines->size(); }
00393
00399   int write_to_disk(const std::string& file_name);
00400
00405   void mpisend(const mixMPI *mpidata);
00406 };
00407 #endif
00408
00409
```

9.17 np_tmcode/src/include/inclu_subs.h File Reference

C++ porting of INCLU functions and subroutines.

Functions

- void cnf (int n, dcomplex z, int nm, dcomplex *csj, dcomplex *csy)

  C++ porting of CNF.
- void exma (dcomplex **am, ParticleDescriptor *c1)

  C++ porting of EXMA.
- void incms (dcomplex **am, double enti, ParticleDescriptor *c1)

  C++ porting of INCMS.
- void indme (int i, int npnt, int npntts, double vk, dcomplex ent, double enti, dcomplex entn, int &jer, int &lcalc, dcomplex &arg, ParticleDescriptor *c1)

  C++ porting of INDME.
- void instr (double **rcf, ParticleDescriptor *c1)

  C++ porting of INSTR.
- void ospv (ParticleDescriptor *c1, double vk, double sze, double exri, dcomplex entn, double enti, int &jer, int &lcalc, dcomplex &arg)

  C++ porting of OSPV.

9.17.1 Detailed Description

This library includes a collection of functions that are used to solve the scattering problem in the case of a sphere with a cluster of inclusions. Like the other use cases, many of the functions declared here execute various calculations on different data structures. In order to manage access to such variety of calculations, most functions are declared as `void` and they operate on output arguments passed by reference.

9.17.2 Function Documentation

9.17.2.1 cnf()

```
void cnf (
            int n,
            dcomplex z,
            int nm,
            dcomplex * csj,
            dcomplex * csy )
```

Parameters

n	`int` Bessel y function order.
z	`dcomplex` Argument of Bessel y function.
nm	`int` Maximum computed order.
csj	`dcomplex *` ANNOTATION: Complex spherical J vector.
csy	`dcomplex *` Complex spherical Bessel functions up to desired order.

9.17.2.2 exma()

```
void exma (
            dcomplex ** am,
            ParticleDescriptor * c1 )
```

Parameters

am	`dcomplex **` Field transition coefficients matrix.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.17.2.3 incms()

```
void incms (
            dcomplex ** am,
            double enti,
            ParticleDescriptor * c1 )
```

Parameters

am	`dcomplex **` Field transition coefficients matrix.
enti	`double` ANNOTATION: imaginary part of coating dielectric function.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.17.2.4 indme()

```
void indme (
            int i,
            int npnt,
            int npntts,
            double vk,
            dcomplex ent,
            double enti,
            dcomplex entn,
            int & jer,
            int & lcalc,
            dcomplex & arg,
            ParticleDescriptor * c1 )
```

Parameters

i	`int` 1-based sphere configuration index.
npnt	`int` ANNOTATION: Number of non transition layer integration points.
npntts	`int` ANNOTATION: Number of transition layer integrtion points.
vk	`double` Vacuum wave vector magnitude.
ent	`dcomplex` ANNOTATION: coating dielectric function.
enti	`double` ANNOTATION: imaginary part of coating dielectric function.
entn	`dcomplex` ANNOTATION: coating refractive index.
jer	`int &` Error code flag.
lcalc	`int &` Maximum order achieved in calculation.
arg	`dcomplex &` ANNOTATION: argument of calculation.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.17.2.5 instr()

```
void instr (
            double ** rcf,
            ParticleDescriptor * c1 )
```

Parameters

rcf	`double **` Pointer to matrix of fractional radii.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.17.2.6 ospv()

```
void ospv (
            ParticleDescriptor * c1,
            double vk,
            double sze,
            double exri,
            dcomplex entn,
            double enti,
            int & jer,
            int & lcalc,
            dcomplex & arg )
```

Parameters

c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.
vk	`double` ANNOTATION: vacuum wave number.
sze	`double` ANNOTATION: size.
exri	`double` External medium refractive index.
entn	`dcomplex` Outer sphere refractive index.
enti	`double` Imaginary part of the outer medium refractive index.
jer	`int &` Reference to an integer error flag.
lcalc	`int &` Maximum order achieved in calculation.
arg	`dcomplex` Complex Bessel function argument.

9.18 inclu_subs.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00029 #ifndef INCLUDE_INCLU_SUBS_H_
00030 #define INCLUDE_INCLU_SUBS_H_
00031
00040 void cnf(int n, dcomplex z, int nm, dcomplex *csj, dcomplex *csy);
00041
00047 void exma(dcomplex **am, ParticleDescriptor *c1);
00048
00055 void incms(dcomplex **am, double enti, ParticleDescriptor *c1);
00056
00071 void indme(
00072        int i, int npnt, int npntts, double vk, dcomplex ent, double enti,
00073        dcomplex entn, int &jer, int &lcalc, dcomplex &arg, ParticleDescriptor *c1
00074 );
00075
00081 void instr(double **rcf, ParticleDescriptor *c1);
00082
00095 void ospv(ParticleDescriptor *c1, double vk, double sze, double exri, dcomplex entn, double enti, int
      &jer, int &lcalc, dcomplex &arg);
00096
00097 #endif // INCLUDE_INCLU_SUBS_H_
```

9.19 np_tmcode/src/include/IterationData.h File Reference

Multi-process communication data structures.

Classes

- class ClusterIterationData

  A data structure representing the information used for a single scale of the CLUSTER case.
- class InclusionIterationData

  A data structure representing the information used for a single scale of the INCLUSION case.
- class SphereIterationData

  A data structure representing the information used for a single scale of the SPHERE case.

9.20 IterationData.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00023 #ifndef INCLUDE_ITERATION_DATA_H_
00024 #define INCLUDE_ITERATION_DATA_H_
00025
00026 // »> DEFINITION OF ClusterIterationData CLASS «<
00030 class ClusterIterationData {
00031 public:
00033   ParticleDescriptor *cl;
00035   double *gaps;
00037   double **tqse;
00039   dcomplex **tqspe;
00041   double **tqss;
00043   dcomplex **tqsps;
00045   double ****zpv;
00047   double **gapm;
00049   dcomplex **gappm;
00051   double *argi;
00053   double *args;
00055   double **gap;
00057   dcomplex **gapp;
00059   double **tqce;
00061   dcomplex **tqcpe;
00063   double **tqcs;
00065   dcomplex **tqcps;
00067   double *duk;
00069   double **cextlr;
00071   double **cext;
00073   double **cmullr;
00075   double **cmul;
00077   double *gapv;
00079   double *tqev;
00081   double *tqsv;
00083   double *u;
00085   double *us;
00087   double *un;
00089   double *uns;
00091   double *up;
00093   double *ups;
00095   double *unmp;
00097   double *unsmp;
00099   double *upmp;
00101   double *upsmp;
00103   double scan;
00105   double cfmp;
00107   double sfmp;
00109   double cfsp;
00111   double sfsp;
00113   double sqsfi;
00115   dcomplex *am_vector;
00117   dcomplex **am;
00119   dcomplex arg;
00121   double vk;
00123   double wn;
00125   double xip;
00127   int number_of_scales;
00129   int xiblock;
00131   int firstxi;
00133   int lastxi;
00135   int proc_device;
00137   int refinemode;
00139   bool is_first_scale;
00141   int maxrefiters;
00143   double accuracygoal;
00144
00152   ClusterIterationData(GeometryConfiguration *gconf, ScattererConfiguration *sconf, const mixMPI
      *mpidata, const int device_count);
00153
00158   ClusterIterationData(const ClusterIterationData& rhs);
00159
00160 #ifdef MPI_VERSION
00166   ClusterIterationData(const mixMPI *mpidata, const int device_count);
00167
00177   void mpibcast(const mixMPI *mpidata);
00178 #endif // MPI_VERSION
00179
00182   ~ClusterIterationData();
00183
00190   static long get_size(GeometryConfiguration *gconf, ScattererConfiguration *sconf);
00191
00199   int update_orders(double** rcf, int inner_order, int outer_order);
00200 };
```

```
00201 // »> END OF ClusterIterationData CLASS DEFINITION «<
00202
00203 // »> DEFINITION OF InclusionIterationData CLASS «<
00207 class InclusionIterationData {
00208 protected:
00210   double *vec_zpv;
00211
00212 public:
00214   int nimd;
00216   double extr;
00217
00219   ParticleDescriptor *cl;
00221   double *gaps;
00223   double **tqse;
00225   dcomplex **tqspe;
00227   double **tqss;
00229   dcomplex **tqsps;
00231   double ****zpv;
00233   double **gapm;
00235   dcomplex **gappm;
00237   double *argi;
00239   double *args;
00241   double **gap;
00243   dcomplex **gapp;
00245   double **tqce;
00247   dcomplex **tqcpe;
00249   double **tqcs;
00251   dcomplex **tqcps;
00253   double *duk;
00255   double **cextlr;
00257   double **cext;
00259   double **cmullr;
00261   double **cmul;
00263   double *gapv;
00265   double *tqev;
00267   double *tqsv;
00269   double *u;
00271   double *us;
00273   double *un;
00275   double *uns;
00277   double *up;
00279   double *ups;
00281   double *unmp;
00283   double *unsmp;
00285   double *upmp;
00287   double *upsmp;
00289   double scan;
00291   double cfmp;
00293   double sfmp;
00295   double cfsp;
00297   double sfsp;
00299   double sqsfi;
00301   dcomplex *am_vector;
00303   dcomplex **am;
00305   dcomplex arg;
00307   double vk;
00309   double wn;
00311   double xip;
00313   int number_of_scales;
00315   int xiblock;
00317   int firstxi;
00319   int lastxi;
00321   int proc_device;
00323   int refinemode;
00325   bool is_first_scale;
00327   int maxrefiters;
00329   double accuracygoal;
00330
00338   InclusionIterationData(GeometryConfiguration *gconf, ScattererConfiguration *sconf, const mixMPI
      *mpidata, const int device_count);
00339
00344   InclusionIterationData(const InclusionIterationData& rhs);
00345
00346 #ifdef MPI_VERSION
00352   InclusionIterationData(const mixMPI *mpidata, const int device_count);
00353
00363   void mpibcast(const mixMPI *mpidata);
00364 #endif
00365
00368   ~InclusionIterationData();
00369
00376   static long get_size(GeometryConfiguration *gconf, ScattererConfiguration *sconf);
00377
00385   int update_orders(double** rcf, int inner_order, int outer_order);
00386 };
00387 // »> END OF InclusionIterationData CLASS DEFINITION «< //
00388
```

```
00389 // »> DEFINITION OF SphereIterationData CLASS «<
00393 class SphereIterationData {
00394 protected:
00396   int _nsph;
00398   int _lm;
00400   double *vec_cmul;
00402   double *vec_cmullr;
00404   dcomplex *vec_tqspe;
00406   dcomplex *vec_tqsps;
00408   double *vec_tqse;
00410   double *vec_tqss;
00412   double *vec_zpv;
00413
00414 public:
00416   double cost;
00418   double sint;
00420   double cosp;
00422   double sinp;
00424   double costs;
00426   double sints;
00428   double cosps;
00430   double sinps;
00432   double vk;
00434   double wn;
00436   double xip;
00438   int number_of_scales;
00440   int xiblock;
00442   int firstxi;
00444   int lastxi;
00446   dcomplex arg;
00448   dcomplex s0;
00450   dcomplex tfsas;
00452   ParticleDescriptor *cl;
00454   double *argi;
00456   double *args;
00458   double scan;
00460   double cfmp;
00462   double sfmp;
00464   double cfsp;
00466   double sfsp;
00468   double *gaps;
00470   double *duk;
00472   double *u;
00474   double *us;
00476   double *un;
00478   double *uns;
00480   double *up;
00482   double *ups;
00484   double *upmp;
00486   double *upsmp;
00488   double *unmp;
00490   double *unsmp;
00492   double **cmul;
00494   double **cmullr;
00496   dcomplex **tqspe;
00498   dcomplex **tqsps;
00500   double **tqse;
00502   double **tqss;
00504   double ****zpv;
00506   bool is_first_scale;
00507
00515   SphereIterationData(GeometryConfiguration *gconf, ScattererConfiguration *sconf, const mixMPI
      *mpidata, const int device_count);
00516
00521   SphereIterationData(const SphereIterationData& rhs);
00522
00523 #ifdef MPI_VERSION
00529   SphereIterationData(const mixMPI *mpidata, const int device_count);
00530
00540   int mpibcast(const mixMPI *mpidata);
00541 #endif // MPI_VERSION
00542
00545   ~SphereIterationData();
00546
00552   int update_order(int order);
00553 };
00554 // »> END OF SphereIterationData CLASS DEFINITION «<
00555
00556 #endif // INCLUDE_ITERATION_DATA_H_
```

9.21 np_tmcode/src/include/lapack_calls.h File Reference

C++ interface to LAPACK calls.

Functions

- void zinvert (dcomplex **mat, np_int n, int &jer)

  Invert a complex matrix with double precision elements.
- void zinvert_and_refine (dcomplex **mat, np_int n, int &jer, int &maxrefiters, double &accuracygoal, int refinemode)

  Invert a complex matrix with double precision elements, using iterative refinement to improve accuracy.

9.21.1 Function Documentation

9.21.1.1 zinvert()

```
void zinvert (
            dcomplex ** mat,
            np_int n,
            int & jer )
```

Use LAPACKE64 to perform an in-place matrix inversion for a complex matrix with double precision elements.

Parameters

mat	Matrix of complex. The matrix to be inverted.
n	`np_int` The number of rows and columns of the [n x n] matrix.
jer	`int &` Reference to an integer return flag.

9.21.1.2 zinvert_and_refine()

```
void zinvert_and_refine (
            dcomplex ** mat,
            np_int n,
            int & jer,
            int & maxrefiters,
            double & accuracygoal,
            int refinemode )
```

Use LAPACKE64 to perform an in-place matrix inversion for a complex matrix with double precision elements.

Parameters

mat	Matrix of complex. The matrix to be inverted.
n	`np_int` The number of rows and columns of the [n x n] matrix.
jer	`int &` Reference to an integer return flag.
maxrefiters	`int` Maximum number of refinement iterations.
accuracygoal	`double &` Accuracy to achieve in iterative refinement, defined as the module of the maximum difference between the identity matrix and the matrix product of the (approximate) inverse times the original matrix. On return, it contains the actually achieved accuracy.
refinemode	`int` Flag to control the refinement mode.

9.22 lapack_calls.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00023 #ifndef INCLUDE_LAPACK_CALLS_H_
00024 #define INCLUDE_LAPACK_CALLS_H_
00025
00035 void zinvert(dcomplex **mat, np_int n, int &jer);
00036
00049 void zinvert_and_refine(dcomplex **mat, np_int n, int &jer, int &maxrefiters, double &accuracygoal,
      int refinemode);
00050
00051 #endif
```

9.23 np_tmcode/src/include/List.h File Reference

A library of classes used to manage dynamic lists.

Classes

- class List< T >

  A class to represent dynamic lists.
- struct List< T >::element

  List element connector.

9.24 List.h

Go to the documentation of this file.

```
00001
00006 #ifndef INCLUDE_LIST_H_
00007 #define INCLUDE_LIST_H_
00008
00022 template<class T> class List {
00023 protected:
00024   int size;
00025   struct element {
00026     T value;
00027     element* p_prev;
00028   };
00029   element *current,
00030     *first,
00031     *last;
00032
00033 public:
00052   List(int length = 1) {
00053     size = length;
00054     first = new element;
00055     first->p_prev = NULL;
00056     current = first;
00057     element *p_prev = first;
00058     for (int i = 1; i < size; i++) {
00059       current = new element;
```

```
00060       current->p_prev = p_prev;
00061       p_prev = current;
00062     }
00063     last = current;
00064   }
00065
00068   ~List() {
00069     current = last;
00070     element *old;
00071     while (current->p_prev) {
00072       old = current;
00073       current = old->p_prev;
00074       delete old;
00075     }
00076     delete current;
00077   }
00078
00090   void append(T value) {
00091     element *p_prev = last;
00092     current = new element;
00093     current->value = value;
00094     current->p_prev = p_prev;
00095     last = current;
00096     size++;
00097   }
00098
00108   T get(int index) {
00109     if (index < 0 || index > size - 1) {
00110       throw ListOutOfBoundsException(index, 0, size - 1);
00111     }
00112     current = last;
00113     for (int i = size - 1; i > index; i--) current = current->p_prev;
00114     return current->value;
00115   }
00116
00123   int length() {
00124     return size;
00125   }
00126
00136   void set(int index, T value) {
00137     if (index < 0 || index > size - 1) {
00138       throw ListOutOfBoundsException(index, 0, size - 1);
00139     }
00140     current = last;
00141     for (int i = size - 1; i > index; i--) current = current->p_prev;
00142     current->value = value;
00143   }
00144
00158   T* to_array() {
00159     T *array = new T[size];
00160     current = last;
00161     for (int i = size - 1; i > -1; i--) {
00162       array[i] = current->value;
00163       current = current->p_prev;
00164     }
00165     return array;
00166   }
00167 };
00168
00169 #endif
```

9.25 np_tmcode/src/include/logging.h File Reference

Definition of the logging system.

Classes

- class Logger

  Logger class.

Macros

- #define **LOG_DEBG** 0

  Debug level logging (maximum verbosity).
- #define **LOG_INFO** 1

  Standard information level logging (default).
- #define **LOG_WARN** 2

  Warning level logging (almost quiet).
- #define **LOG_ERRO** 3

  Error level logging (silent, unless in pain).
- #define **TOSTRING**(ARG) string(#ARG)

  Macro to stringize code lines.

9.26 logging.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00021 #ifndef INCLUDE_LOGGING_H_
00022 #define INCLUDE_LOGGING_H_
00023
00025 #define LOG_DEBG 0
00027 #define LOG_INFO 1
00029 #define LOG_WARN 2
00031 #define LOG_ERRO 3
00032
00034 #define TOSTRING(ARG) string(#ARG)
00035
00049 class Logger {
00050  protected:
00052   FILE *err_output;
00054   FILE *log_output;
00056   std::string *last_message;
00058   int log_threshold;
00060   long repetitions;
00061
00062  public:
00074   Logger(int threshold, FILE *logging_output = stdout, FILE *error_output = stderr);
00075
00078   ~Logger();
00079
00084   void err(const std::string& message);
00085
00090   void flush(int level=LOG_DEBG);
00091
00097   void log(const std::string& message, int level=LOG_INFO);
00098
00109   void push(const std::string& message);
00110 };
00111
00112 #endif
```

9.27 np_tmcode/src/include/magma_calls.h File Reference

C++ interface to MAGMA calls.

```
#include <string>
```

Functions

- void magma_zinvert (dcomplex **mat, np_int n, int &jer, int device_id=0)

  Invert a complex matrix with double precision elements.
- void magma_zinvert1 (dcomplex *&inva, np_int n, int &jer, int device_id)

  Auxiliary function to Invert a complex matrix with double precision elements.
- void magma_zinvert_and_refine (dcomplex **mat, np_int n, int &jer, int &maxrefiters, double &accuracygoal, int refinemode, int device_id, const string &output_path, int jxi488)

  Invert a complex matrix with double precision elements, applying iterative refinement of the solution.
- void magma_refine (dcomplex *aorig, dcomplex *inva, np_int n, int &jer, int &maxrefiters, double &accuracygoal, int refinemode, int device_id, const string &output_path, int jxi488)

  Apply iterative refinement of the solution of a matrix inversion.

9.27.1 Function Documentation

9.27.1.1 magma_refine()

```
void magma_refine (
            dcomplex * aorig,
            dcomplex * inva,
            np_int n,
            int & jer,
            int & maxrefiters,
            double & accuracygoal,
            int refinemode,
            int device_id,
            const string & output_path,
            int jxi488 )
```

Iteratively compute and apply a correction to the inverse `inva` of the complex matrix `aorig`, for a maximum number of `maxiters` times, or until achieving a maximum residual better than `accuracygoal`.

Parameters

aorig	pointer to the first element of the matrix of complex to be inverted.
inva	pointer to the first element of inverse.
n	`np_int` The number of rows and columns of the [n x n] matrices.
jer	`int &` Reference to an integer return flag.
maxrefiters	`int` Maximum number of refinement iterations to apply.
accuracygoal	`double` Accuracy to achieve in iterative refinement, defined as the module of the maximum difference between the identity matrix and the matrix product of the (approximate) inverse times the original matrix. On return, it contains the actually achieved accuracy.
refinemode	`int` Flag for refinement mode selection.
device_id	`int` ID of the device for matrix inversion offloading.
output_path	`const string &` Path where the output needs to be placed.
jxi488	`int` Index of the current wavelength calculation.

9.27.1.2 magma_zinvert()

```
void magma_zinvert (
```

```
            dcomplex ** mat,
            np_int n,
            int & jer,
            int device_id = 0 )
```

call magma_zinvert1() to do the actual inversion

Parameters

mat	Matrix of complex. The matrix to be inverted.
n	np_int The number of rows and columns of the [n x n] matrix.
jer	int & Reference to an integer return flag.
device_id	int ID of the device for matrix inversion offloading.

9.27.1.3 magma_zinvert1()

```
void magma_zinvert1 (
            dcomplex *& inva,
            np_int n,
            int & jer,
            int device_id )
```

Use MAGMA to perform an in-place matrix inversion for a complex matrix with double precision elements.

Parameters

inva	reference to the pointer to the first element of the matrix to be inverted on entry, inverted matrix on exit.
n	np_int The number of rows and columns of the [n x n] matrix.
jer	int & Reference to an integer return flag.
device_id	int ID of the device for matrix inversion offloading.

9.27.1.4 magma_zinvert_and_refine()

```
void magma_zinvert_and_refine (
            dcomplex ** mat,
            np_int n,
            int & jer,
            int & maxrefiters,
            double & accuracygoal,
            int refinemode,
            int device_id,
            const string & output_path,
            int jxi488 )
```

call magma_zinvert1() to perform the first matrix inversion, then magma_refine() to do the refinement (only if maxrefiters is >0)

Parameters

mat	Matrix of complex. The matrix to be inverted.
n	`np_int` The number of rows and columns of the [n x n] matrix.
jer	`int &` Reference to an integer return flag.
maxrefiters	`int` Maximum number of refinement iterations to apply.
accuracygoal	`double &` Accuracy to achieve in iterative refinement, defined as the module of the maximum difference between the identity matrix and the matrix product of the (approximate) inverse times the original matrix. On return, it contains the actually achieved accuracy.
refinemode	`int` Flag to control the refinement mode.
device_id	`int` ID of the device for matrix inversion offloading.
output_path	`const string &` Path where the output needs to be placed.
jxi488	`int` Index of the current wavelength calculation.

9.28 magma_calls.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00023 #include <string>
00024
00025 #ifndef INCLUDE_MAGMA_CALLS_H_
00026 #define INCLUDE_MAGMA_CALLS_H_
00027
00037 void magma_zinvert(dcomplex **mat, np_int n, int &jer, int device_id=0);
00038
00049 void magma_zinvert1(dcomplex * &inva, np_int n, int &jer, int device_id);
00050
00065 void magma_zinvert_and_refine(dcomplex **mat, np_int n, int &jer, int &maxrefiters, double
      &accuracygoal, int refinemode, int device_id, const string& output_path, int jxi488);
00066
00084 void magma_refine(dcomplex *aorig, dcomplex *inva, np_int n, int &jer, int &maxrefiters, double
      &accuracygoal, int refinemode, int device_id, const string& output_path, int jxi488);
00085
00086 #endif
```

9.29 np_tmcode/src/include/outputs.h File Reference

Definition of the output format system.

Classes

- class ClusterOutputInfo

  Class to collect output information for scattering from clusters.
- class InclusionOutputInfo

Class to collect output information for scattering from particle with inclusions.

- class SphereOutputInfo

  Class to collect output information for scattering from a single sphere.

- class TrappingOutputInfo

  Class to collect output information for particle trapping.

9.30 outputs.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00021 #ifndef INCLUDE_OUTPUTS_H_
00022 #define INCLUDE_OUTPUTS_H_
00023
00024 // »» OUTPUT FOR CLUSTER ««
00034 class ClusterOutputInfo {
00035 protected:
00037   int _skip_flag;
00039   int _num_theta;
00041   int _num_thetas;
00043   int _num_phi;
00045   int _num_phis;
00047   int _first_xi;
00048
00054   int write_hdf5(const std::string &file_name);
00055
00065   int write_legacy(const std::string &output);
00066
00067 public:
00069   const int &skip_flag = _skip_flag;
00071   const int &first_xi = _first_xi;
00073   int nsph;
00075   int li;
00077   int le;
00079   int lm;
00081   np_int mxndm;
00083   int inpol;
00085   int npnt;
00087   int npntts;
00089   int iavm;
00091   int isam;
00093   int idfc;
00095   double *vec_x_coords;
00097   double *vec_y_coords;
00099   double *vec_z_coords;
00101   double th;
00103   double thstp;
00105   double thlst;
00107   double ths;
00109   double thsstp;
00111   double thslst;
00113   double ph;
00115   double phstp;
00117   double phlst;
00119   double phs;
00121   double phsstp;
00123   double phslst;
00125   int ndirs;
00127   double exri;
00129   int nxi;
00131   int xi_block_size;
00133   int jwtm;
00135   int *vec_jxi;
00137   short *vec_ier;
```

```
00139   double *vec_vk;
00141   double *vec_xi;
00143   int configurations;
00145   double *vec_sphere_sizes;
00147   dcomplex *vec_sphere_ref_indices;
00149   double *vec_sphere_scs;
00151   double *vec_sphere_abs;
00153   double *vec_sphere_exs;
00155   double *vec_sphere_albs;
00157   double *vec_sphere_sqscs;
00159   double *vec_sphere_sqabs;
00161   double *vec_sphere_sqexs;
00163   dcomplex *vec_fsas;
00165   double *vec_qschus;
00167   double *vec_pschus;
00169   double *vec_s0mags;
00171   double *vec_cosavs;
00173   double *vec_raprs;
00175   double *vec_tqek1;
00177   double *vec_tqsk1;
00179   double *vec_tqek2;
00181   double *vec_tqsk2;
00183   dcomplex *vec_fsat;
00185   double *vec_qschut;
00187   double *vec_pschut;
00189   double *vec_s0magt;
00191   double tgs;
00193   double *vec_scc1;
00195   double *vec_scc2;
00197   double *vec_abc1;
00199   double *vec_abc2;
00201   double *vec_exc1;
00203   double *vec_exc2;
00205   double *vec_albedc1;
00207   double *vec_albedc2;
00209   double *vec_qscamc1;
00211   double *vec_qscamc2;
00213   double *vec_qabsmc1;
00215   double *vec_qabsmc2;
00217   double *vec_qextmc1;
00219   double *vec_qextmc2;
00221   double *vec_sccrt1;
00223   double *vec_sccrt2;
00225   double *vec_abcrt1;
00227   double *vec_abcrt2;
00229   double *vec_excrt1;
00231   double *vec_excrt2;
00233   dcomplex *vec_fsac11;
00235   dcomplex *vec_fsac21;
00237   dcomplex *vec_fsac22;
00239   dcomplex *vec_fsac12;
00241   double *vec_qschuc1;
00243   double *vec_qschuc2;
00245   double *vec_pschuc1;
00247   double *vec_pschuc2;
00249   double *vec_s0magc1;
00251   double *vec_s0magc2;
00253   double *vec_cosavc1;
00255   double *vec_cosavc2;
00257   double *vec_raprc1;
00259   double *vec_raprc2;
00261   double *vec_fkc1;
00263   double *vec_fkc2;
00265   double *vec_dir_tidg;
00267   double *vec_dir_pidg;
00269   double *vec_dir_tsdg;
00271   double *vec_dir_psdg;
00273   double *vec_dir_scand;
00275   double *vec_dir_cfmp;
00277   double *vec_dir_sfmp;
00279   double *vec_dir_cfsp;
00281   double *vec_dir_sfsp;
00283   double *vec_dir_un;
00285   double *vec_dir_uns;
00287   dcomplex *vec_dir_sas11;
00289   dcomplex *vec_dir_sas21;
00291   dcomplex *vec_dir_sas12;
00293   dcomplex *vec_dir_sas22;
00295   double *vec_dir_muls;
00297   double *vec_dir_mulslr;
00299   dcomplex *vec_dir_sat11;
00301   dcomplex *vec_dir_sat21;
00303   dcomplex *vec_dir_sat12;
00305   dcomplex *vec_dir_sat22;
00307   double *vec_dir_scc1;
00309   double *vec_dir_scc2;
00311   double *vec_dir_abc1;
```

```
00313   double *vec_dir_abc2;
00315   double *vec_dir_exc1;
00317   double *vec_dir_exc2;
00319   double *vec_dir_albedc1;
00321   double *vec_dir_albedc2;
00323   double *vec_dir_qscc1;
00325   double *vec_dir_qscc2;
00327   double *vec_dir_qabc1;
00329   double *vec_dir_qabc2;
00331   double *vec_dir_qexc1;
00333   double *vec_dir_qexc2;
00335   double *vec_dir_sccrt1;
00337   double *vec_dir_sccrt2;
00339   double *vec_dir_abcrt1;
00341   double *vec_dir_abcrt2;
00343   double *vec_dir_excrt1;
00345   double *vec_dir_excrt2;
00347   dcomplex *vec_dir_fsac11;
00349   dcomplex *vec_dir_fsac21;
00351   dcomplex *vec_dir_fsac12;
00353   dcomplex *vec_dir_fsac22;
00355   dcomplex *vec_dir_sac11;
00357   dcomplex *vec_dir_sac21;
00359   dcomplex *vec_dir_sac12;
00361   dcomplex *vec_dir_sac22;
00363   double *vec_dir_qschuc1;
00365   double *vec_dir_qschuc2;
00367   double *vec_dir_pschuc1;
00369   double *vec_dir_pschuc2;
00371   double *vec_dir_s0magc1;
00373   double *vec_dir_s0magc2;
00375   double *vec_dir_cosavc1;
00377   double *vec_dir_cosavc2;
00379   double *vec_dir_raprc1;
00381   double *vec_dir_raprc2;
00383   double *vec_dir_flc1;
00385   double *vec_dir_flc2;
00387   double *vec_dir_frc1;
00389   double *vec_dir_frc2;
00391   double *vec_dir_fkc1;
00393   double *vec_dir_fkc2;
00395   double *vec_dir_fxc1;
00397   double *vec_dir_fxc2;
00399   double *vec_dir_fyc1;
00401   double *vec_dir_fyc2;
00403   double *vec_dir_fzc1;
00405   double *vec_dir_fzc2;
00407   double *vec_dir_tqelc1;
00409   double *vec_dir_tqelc2;
00411   double *vec_dir_tqerc1;
00413   double *vec_dir_tqerc2;
00415   double *vec_dir_tqekc1;
00417   double *vec_dir_tqekc2;
00419   double *vec_dir_tqexc1;
00421   double *vec_dir_tqexc2;
00423   double *vec_dir_tqeyc1;
00425   double *vec_dir_tqeyc2;
00427   double *vec_dir_tqezc1;
00429   double *vec_dir_tqezc2;
00431   double *vec_dir_tqslc1;
00433   double *vec_dir_tqslc2;
00435   double *vec_dir_tqsrc1;
00437   double *vec_dir_tqsrc2;
00439   double *vec_dir_tqskc1;
00441   double *vec_dir_tqskc2;
00443   double *vec_dir_tqsxc1;
00445   double *vec_dir_tqsxc2;
00447   double *vec_dir_tqsyc1;
00449   double *vec_dir_tqsyc2;
00451   double *vec_dir_tqszc1;
00453   double *vec_dir_tqszc2;
00455   double *vec_dir_mulc;
00457   double *vec_dir_mulclr;
00458
00467   ClusterOutputInfo(
00468     ScattererConfiguration *sc, GeometryConfiguration *gc,
00469     const mixMPI *mpidata, int first_xi = 1, int xi_length = 0
00470   );
00471
00476   ClusterOutputInfo(const std::string &hdf5_name);
00477
00482   ClusterOutputInfo(const int skip_flag);
00483
00486   ~ClusterOutputInfo();
00487
00496   static long compute_size(
00497     ScattererConfiguration *sc, GeometryConfiguration *gc,
```

```
00498     int first_xi = 1, int xi_length = 0
00499   );
00500
00505   long compute_size();
00506
00512   int insert(const ClusterOutputInfo &rhs);
00513
00520   int write(const std::string &output, const std::string &format);
00521
00522 #ifdef MPI_VERSION
00534   int mpireceive(const mixMPI* mpidata, int pid);
00535
00547   int mpisend(const mixMPI *mpidata);
00548 #endif // MPI_VERSION
00549 };
00550 // »> END OF OUTPUT FOR CLUSTER «<
00551
00552 // »> OUTPUT FOR INCLUSION «<
00562 class InclusionOutputInfo {
00563 protected:
00565   int _skip_flag;
00567   int _num_theta;
00569   int _num_thetas;
00571   int _num_phi;
00573   int _num_phis;
00575   int _first_xi;
00576
00582   int write_hdf5(const std::string &file_name);
00583
00593   int write_legacy(const std::string &output);
00594
00595 public:
00597   const int &skip_flag = _skip_flag;
00599   const int &first_xi = _first_xi;
00601   int nsph;
00603   int li;
00605   int le;
00607   int lm;
00609   np_int mxndm;
00611   int inpol;
00613   int npnt;
00615   int npntts;
00617   int iavm;
00619   int isam;
00621   int idfc;
00623   double *vec_x_coords;
00625   double *vec_y_coords;
00627   double *vec_z_coords;
00629   double th;
00631   double thstp;
00633   double thlst;
00635   double ths;
00637   double thsstp;
00639   double thslst;
00641   double ph;
00643   double phstp;
00645   double phlst;
00647   double phs;
00649   double phsstp;
00651   double phslst;
00653   int ndirs;
00655   double exri;
00657   int nxi;
00659   int xi_block_size;
00661   int jwtm;
00663   int *vec_jxi;
00665   short *vec_ier;
00667   double *vec_vk;
00669   double *vec_xi;
00671   int configurations;
00673   double *vec_sphere_sizes;
00675   dcomplex *vec_sphere_ref_indices;
00677   double *vec_scs1;
00679   double *vec_scs2;
00681   double *vec_abs1;
00683   double *vec_abs2;
00685   double *vec_exs1;
00687   double *vec_exs2;
00689   double *vec_albeds1;
00691   double *vec_albeds2;
00693   double *vec_scsrt1;
00695   double *vec_scsrt2;
00697   double *vec_absrt1;
00699   double *vec_absrt2;
00701   double *vec_exsrt1;
00703   double *vec_exsrt2;
00705   double *vec_qschu1;
```

```
00707   double *vec_qschu2;
00709   double *vec_pschu1;
00711   double *vec_pschu2;
00713   double *vec_s0mag1;
00715   double *vec_s0mag2;
00717   double *vec_cosav1;
00719   double *vec_cosav2;
00721   double *vec_raprs1;
00723   double *vec_raprs2;
00725   double *vec_fk1;
00727   double *vec_fk2;
00729   dcomplex *vec_fsas11;
00731   dcomplex *vec_fsas21;
00733   dcomplex *vec_fsas22;
00735   dcomplex *vec_fsas12;
00737   double *vec_dir_tidg;
00739   double *vec_dir_pidg;
00741   double *vec_dir_tsdg;
00743   double *vec_dir_psdg;
00745   double *vec_dir_scand;
00747   double *vec_dir_cfmp;
00749   double *vec_dir_sfmp;
00751   double *vec_dir_cfsp;
00753   double *vec_dir_sfsp;
00755   double *vec_dir_un;
00757   double *vec_dir_uns;
00759   double *vec_dir_scs1;
00761   double *vec_dir_scs2;
00763   double *vec_dir_abs1;
00765   double *vec_dir_abs2;
00767   double *vec_dir_exs1;
00769   double *vec_dir_exs2;
00771   double *vec_dir_albeds1;
00773   double *vec_dir_albeds2;
00775   double *vec_dir_scsrt1;
00777   double *vec_dir_scsrt2;
00779   double *vec_dir_absrt1;
00781   double *vec_dir_absrt2;
00783   double *vec_dir_exsrt1;
00785   double *vec_dir_exsrt2;
00787   dcomplex *vec_dir_fsas11;
00789   dcomplex *vec_dir_fsas21;
00791   dcomplex *vec_dir_fsas12;
00793   dcomplex *vec_dir_fsas22;
00795   dcomplex *vec_dir_sas11;
00797   dcomplex *vec_dir_sas21;
00799   dcomplex *vec_dir_sas12;
00801   dcomplex *vec_dir_sas22;
00803   double *vec_dir_qschu1;
00805   double *vec_dir_qschu2;
00807   double *vec_dir_pschu1;
00809   double *vec_dir_pschu2;
00811   double *vec_dir_s0mag1;
00813   double *vec_dir_s0mag2;
00815   double *vec_dir_cosav1;
00817   double *vec_dir_cosav2;
00819   double *vec_dir_rapr1;
00821   double *vec_dir_rapr2;
00823   double *vec_dir_fl1;
00825   double *vec_dir_fl2;
00827   double *vec_dir_fr1;
00829   double *vec_dir_fr2;
00831   double *vec_dir_fk1;
00833   double *vec_dir_fk2;
00835   double *vec_dir_fx1;
00837   double *vec_dir_fx2;
00839   double *vec_dir_fy1;
00841   double *vec_dir_fy2;
00843   double *vec_dir_fz1;
00845   double *vec_dir_fz2;
00847   double *vec_dir_tqel1;
00849   double *vec_dir_tqel2;
00851   double *vec_dir_tqer1;
00853   double *vec_dir_tqer2;
00855   double *vec_dir_tqek1;
00857   double *vec_dir_tqek2;
00859   double *vec_dir_tqex1;
00861   double *vec_dir_tqex2;
00863   double *vec_dir_tqey1;
00865   double *vec_dir_tqey2;
00867   double *vec_dir_tqez1;
00869   double *vec_dir_tqez2;
00871   double *vec_dir_tqsl1;
00873   double *vec_dir_tqsl2;
00875   double *vec_dir_tqsr1;
00877   double *vec_dir_tqsr2;
00879   double *vec_dir_tqsk1;
```

```
00881   double *vec_dir_tqsk2;
00883   double *vec_dir_tqsx1;
00885   double *vec_dir_tqsx2;
00887   double *vec_dir_tqsy1;
00889   double *vec_dir_tqsy2;
00891   double *vec_dir_tqsz1;
00893   double *vec_dir_tqsz2;
00895   double *vec_dir_mull;
00897   double *vec_dir_mulllr;
00898
00907   InclusionOutputInfo(
00908     ScattererConfiguration *sc, GeometryConfiguration *gc,
00909     const mixMPI *mpidata, int first_xi = 1, int xi_length = 0
00910   );
00911
00916   InclusionOutputInfo(const std::string &hdf5_name);
00917
00922   InclusionOutputInfo(const int skip_flag);
00923
00926   ~InclusionOutputInfo();
00927
00936   static long compute_size(
00937     ScattererConfiguration *sc, GeometryConfiguration *gc,
00938     int first_xi = 1, int xi_length = 0
00939   );
00940
00945   long compute_size();
00946
00952   int insert(const InclusionOutputInfo &rhs);
00953
00960   int write(const std::string &output, const std::string &format);
00961
00962 #ifdef MPI_VERSION
00974   int mpireceive(const mixMPI* mpidata, int pid);
00975
00987   int mpisend(const mixMPI *mpidata);
00988 #endif // MPI_VERSION
00989 };
00990 // »> END OF OUTPUT FOR INCLUSION «<
00991
00992 // »> OUTPUT FOR SPHERE «<
01002 class SphereOutputInfo {
01003 protected:
01005   int _skip_flag;
01007   int _num_theta;
01009   int _num_thetas;
01011   int _num_phi;
01013   int _num_phis;
01015   int _first_xi;
01016
01022   int write_hdf5(const std::string &file_name);
01023
01033   int write_legacy(const std::string &output);
01034
01035 public:
01037   const int &skip_flag = _skip_flag;
01039   const int &first_xi = _first_xi;
01041   int nsph;
01043   int lm;
01045   int inpol;
01047   int npnt;
01049   int npntts;
01051   int isam;
01053   int idfc;
01055   double th;
01057   double thstp;
01059   double thlst;
01061   double ths;
01063   double thsstp;
01065   double thslst;
01067   double ph;
01069   double phstp;
01071   double phlst;
01073   double phs;
01075   double phsstp;
01077   double phslst;
01079   int ndirs;
01081   double exri;
01083   int nxi;
01085   int xi_block_size;
01087   int jwtm;
01089   int configurations;
01091   int lcalc;
01093   dcomplex arg;
01095   int *vec_jxi;
01097   short *vec_ier;
01099   double *vec_vk;
```

```
01101   double *vec_xi;
01103   double *vec_sphere_sizes;
01105   dcomplex *vec_sphere_ref_indices;
01107   double *vec_scs;
01109   double *vec_abs;
01111   double *vec_exs;
01113   double *vec_albeds;
01115   double *vec_scsrt;
01117   double *vec_absrt;
01119   double *vec_exsrt;
01121   dcomplex *vec_fsas;
01123   double *vec_qschu;
01125   double *vec_pschu;
01127   double *vec_s0mag;
01129   double *vec_cosav;
01131   double *vec_raprs;
01133   double *vec_tqek1;
01135   double *vec_tqek2;
01137   double *vec_tqsk1;
01139   double *vec_tqsk2;
01141   dcomplex *vec_fsat;
01143   double *vec_qschut;
01145   double *vec_pschut;
01147   double *vec_s0magt;
01149   double *vec_dir_tidg;
01151   double *vec_dir_pidg;
01153   double *vec_dir_tsdg;
01155   double *vec_dir_psdg;
01157   double *vec_dir_scand;
01159   double *vec_dir_cfmp;
01161   double *vec_dir_sfmp;
01163   double *vec_dir_cfsp;
01165   double *vec_dir_sfsp;
01167   double *vec_dir_un;
01169   double *vec_dir_uns;
01171   dcomplex *vec_dir_sas11;
01173   dcomplex *vec_dir_sas21;
01175   dcomplex *vec_dir_sas12;
01177   dcomplex *vec_dir_sas22;
01179   double *vec_dir_fx;
01181   double *vec_dir_fy;
01183   double *vec_dir_fz;
01185   double *vec_dir_muls;
01187   double *vec_dir_mulslr;
01188
01197   SphereOutputInfo(
01198     ScattererConfiguration *sc, GeometryConfiguration *gc,
01199     const mixMPI *mpidata, int first_xi = 1, int xi_length = 0
01200   );
01201
01206   SphereOutputInfo(const std::string &hdf5_name);
01207
01212   SphereOutputInfo(const int skip_flag);
01213
01216   ~SphereOutputInfo();
01217
01226   static long compute_size(
01227     ScattererConfiguration *sc, GeometryConfiguration *gc,
01228     int first_xi = 1, int xi_length = 0
01229   );
01230
01235   long compute_size();
01236
01242   int insert(const SphereOutputInfo &rhs);
01243
01250   int write(const std::string &output, const std::string &format);
01251
01252 #ifdef MPI_VERSION
01264   int mpireceive(const mixMPI *mpidata, int pid);
01265
01277   int mpisend(const mixMPI *mpidata);
01278 #endif // MPI_VERSION
01279 };
01280 // »> END OF OUTPUT FOR SPHERE «<
01281
01282 // »> OUTPUT FOR TRAPPING «<
01292 class TrappingOutputInfo {
01293 protected:
01295   int _jft;
01297   int _jss;
01299   int _jtw;
01301   int _nkv;
01303   int _nxv;
01305   int _nyv;
01307   int _nzv;
01309   int _lmode;
01311   int _le;
```

```
01313   double _vk;
01315   double _exri;
01317   double _an;
01319   double _ff;
01321   double _tra;
01323   double _spd;
01325   double _frsh;
01327   double _exril;
01328
01334   int write_hdf5(const std::string &file_name);
01335
01345   int write_legacy(const std::string &output);
01346
01347 public:
01349   const int& jft = _jft;
01351   const int& jss = _jss;
01353   const int& jtw = _jtw;
01355   const int& nkv = _nkv;
01357   const int& nxv = _nxv;
01359   const int& nyv = _nyv;
01361   const int& nzv = _nzv;
01363   const int& lmode = _lmode;
01365   const int& le = _le;
01367   const double& vk = _vk;
01369   const double& exri = _exri;
01371   const double& an = _an;
01373   const double& ff = _ff;
01375   const double& tra = _tra;
01377   const double& spd = _spd;
01379   const double& frsh = _frsh;
01381   const double& exril = _exril;
01383   double *vec_x;
01385   double *vec_y;
01387   double *vec_z;
01389   double *vec_csf1;
01391   double *vec_csf2;
01393   double *vec_csf3;
01395   double *vec_cst1;
01397   double *vec_cst2;
01399   double *vec_cst3;
01400
01410   TrappingOutputInfo(
01411     int ift, int iss, int itw, int nx, int ny, int nz
01412   );
01413
01416   ~TrappingOutputInfo();
01417
01427   static long get_size(int ift, int iss, int nx, int ny, int nz);
01428
01433   long get_size();
01434
01440   void set_param(const std::string& pname, double pvalue);
01441
01448   int write(const std::string &output, const std::string &format);
01449 };
01450 // »> END OF OUTPUT FOR TRAPPING «<
01451
01452 #endif // INCLUDE_OUTPUTS_H_
```

9.31 np_tmcode/src/include/Parsers.h File Reference

A library of functions designed to parse formatted input into memory.

Macros

- #define **FILE_NOT_FOUND_ERROR** 21

  Error code if a file is not found.

Functions

- std::string ∗ load_file (const std::string &file_name, int ∗count)

  Load a text file as a sequence of strings in memory.

9.31.1 Function Documentation

9.31.1.1 load_file()

```
std::string * load_file (
            const std::string & file_name,
            int * count )
```

The configuration of the field expansion code in FORTRAN uses shared memory access and file I/O operations managed by different functions. Although this approach could be theoretically replicated, it is more convenient to handle input and output to distinct files using specific functions. load_file() helps in the task of handling input such as configuration files or text data structures that need to be loaded entirely. The function performs a line-by line scan of the input file and returns an array of strings that can be later parsed and ingested by the concerned code blocks. An optional pointer to integer allows the function to keep track of the number of file lines that were read, if needed.

Parameters

file_name	`string` The path of the file to be read.
count	`int*` Pointer to an integer recording the number of read lines [OPTIONAL, default=NULL].

Returns

array_lines `string*` An array of strings, one for each input file line.

9.32 Parsers.h

Go to the documentation of this file.

```
00001
00007 #ifndef FILE_NOT_FOUND_ERROR
00009 #define FILE_NOT_FOUND_ERROR 21
00010 #endif
00011
00012 #ifndef INCLUDE_PARSERS_H_
00013 #define INCLUDE_PARSERS_H_
00014
00035 std::string *load_file(const std::string& file_name, int *count);
00036
00037 #endif /* INCLUDE_PARSERS_H_ */
```

9.33 np_tmcode/src/include/sph_subs.h File Reference

C++ porting of SPH functions and subroutines.

Functions

- void aps (double ****zpv, int li, int nsph, ParticleDescriptor *c1, double sqk, double *gaps)

  Compute the asymmetry-corrected scattering cross-section of a sphere.
- void cbf (int n, dcomplex z, int &nm, dcomplex *csj)

  Complex Bessel Function.
- double cg1 (int lmpml, int mu, int l, int m)

  Clebsch-Gordan coefficients.

- void diel (int npntmo, int ns, int i, int ic, double vk, ParticleDescriptor *c1)

  Compute the continuous variation of the refractive index and of its radial derivative.
- void dme (int li, int i, int npnt, int npntts, double vk, double exdc, double exri, ParticleDescriptor *c1, int &jer, int &lcalc, dcomplex &arg, int last_conf=0)

  Compute Mie scattering coefficients.
- double envj (int n, double x)

  Bessel function calculation control parameters.
- void mmulc (dcomplex *vint, double **cmullr, double **cmul)

  Compute the Mueller Transformation Matrix.
- int msta1 (double x, int mp)

  Starting point for Bessel function magnitude.
- int msta2 (double x, int n, int mp)

  Starting point for Bessel function precision.
- void orunve (double *u1, double *u2, double *u3, int iorth, double torth)

  Compute the amplitude of the orthogonal unit vector.
- void pwma (double *up, double *un, dcomplex *ylm, int inpol, int lw, int isq, ParticleDescriptor *c1)

  Compute incident and scattered field amplitudes.
- void rabas (int inpol, int li, int nsph, ParticleDescriptor *c1, double **tqse, dcomplex **tqspe, double **tqss, dcomplex **tqsps)

  Compute radiation torques on a single sphere in Cartesian coordinates.
- void rbf (int n, double x, int &nm, double sj[ ])

  Real Bessel Function.
- void rkc (int npntmo, double step, dcomplex dcc, double &x, int lpo, dcomplex &y1, dcomplex &y2, dcomplex &dy1, dcomplex &dy2)

  Soft layer radial function and derivative.
- void rkt (int npntmo, double step, double &x, int lpo, dcomplex &y1, dcomplex &y2, dcomplex &dy1, dcomplex &dy2, ParticleDescriptor *c1)

  Transition layer radial function and derivative.
- void rnf (int n, double x, int &nm, double *sy)

  Spherical Bessel functions.
- void sphar (double cosrth, double sinrth, double cosrph, double sinrph, int ll, dcomplex *ylm)

  Spherical harmonics for given direction.
- void sscr0 (dcomplex &tfsas, int nsph, int lm, double vk, double exri, ParticleDescriptor *c1)

  Compute scattering, absorption and extinction cross-sections.
- void sscr2 (int nsph, int lm, double vk, double exri, ParticleDescriptor *c1)

  C++ Compute the scattering amplitude and the scattered field intensity.
- void thdps (int lm, double ****zpv)

  Determine the geometrical asymmetry parameter coefficients.
- void upvmp (double thd, double phd, int icspnv, double &cost, double &sint, double &cosp, double &sinp, double *u, double *up, double *un)

  Compute the unitary vectors onb the polarization plane and its orthogonal direction.
- void upvsp (double *u, double *upmp, double *unmp, double *us, double *upsmp, double *unsmp, double *up, double *un, double *ups, double *uns, double *duk, int &isq, int &ibf, double &scand, double &cfmp, double &sfmp, double &cfsp, double &sfsp)

  Compute the unitary vector perpendicular to incident and scattering plane.
- void wmamp (int iis, double cost, double sint, double cosp, double sinp, int inpol, int lm, int idot, int nsph, double *arg, double *u, double *up, double *un, ParticleDescriptor *c1)

  Compute meridional plane-referred geometrical asymmetry parameter coefficients.
- void wmasp (double cost, double sint, double cosp, double sinp, double costs, double sints, double cosps, double sinps, double *u, double *up, double *un, double *us, double *ups, double *uns, int isq, int ibf, int inpol, int lm, int idot, int nsph, double *argi, double *args, ParticleDescriptor *c1)

  Compute the scattering plane-referred geometrical asymmetry parameter coefficients.

9.33.1 Detailed Description

This library includes a collection of functions that are used to solve the scattering problem in the case of a single sphere. Some of these functions are also generalized to the case of clusters of spheres. In most cases, the results of calculations do not fall back to fundamental data types. They are rather multi-component structures. In order to manage access to such variety of return values, most functions are declared as `void` and they operate on output arguments passed by reference.

9.33.2 Function Documentation

9.33.2.1 aps()

```
void aps (
            double **** zpv,
            int li,
            int nsph,
            ParticleDescriptor * c1,
            double sqk,
            double * gaps )
```

This function computes the product between the geometrical asymmetry parameter and the scattering cross-section. See Sec. 3.2.1 of Borghese, Denti & Saija (2007).

Parameters

zpv	`double ****` Geometrical asymmetry parameter coefficients matrix.
li	`int` Maximum field expansion order.
nsph	`int` Number of spheres.
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.
sqk	`double`
gaps	`double *` Geometrical asymmetry parameter-corrected cross-section.

9.33.2.2 cbf()

```
void cbf (
            int n,
            dcomplex z,
            int & nm,
            dcomplex * csj )
```

This function computes the complex spherical Bessel funtions j. It uses the auxiliary functions `msta1()` and `msta2()` to determine the starting point for backward recurrence. This is the `CSPHJ` implementation of the `specfun` library.

Parameters

n	`int` Order of the function.
z	`complex double` Argument of the function.
nm	`int &` Highest computed order.
csj	`complex double *` The desired function j.

9.33.2.3 cg1()

```
double cg1 (
            int lmpml,
            int mu,
            int l,
            int m )
```

This function comutes the Clebsch-Gordan coefficients for the irreducible spherical tensors. See Sec. 1.6.3 and Table 1.1 in Borghese, Denti & Saija (2007).

Parameters

lmpml	`int`
mu	`int`
l	`int`
m	`int`

Returns

result: `double` Clebsh-Gordan coefficient.

9.33.2.4 diel()

```
void diel (
            int npntmo,
            int ns,
            int i,
            int ic,
            double vk,
            ParticleDescriptor * c1 )
```

This function implements the continuous variation of the refractive index and of its radial derivative through the materials that constitute the sphere and its surrounding medium. See Sec. 5.2 in Borghese, Denti & Saija (2007).

Parameters

npntmo	`int`
ns	`int`
i	`int`
ic	`int`
vk	`double`
c1	`ParticleDescriptor` * Pointer to a `ParticleDescriptor` data structure.

9.33.2.5 dme()

```
void dme (
            int li,
            int i,
```

```
            int npnt,
            int npntts,
            double vk,
            double exdc,
            double exri,
            ParticleDescriptor * c1,
            int & jer,
            int & lcalc,
            dcomplex & arg,
            int last_conf = 0 )
```

This function determines the L-dependent Mie scattering coefficients a_l and b_l for the cases of homogeneous spheres, radially non-homogeneous spheres and, in case of sphere with dielectric function, sustaining longitudinal waves. See Sec. 5.1 in Borghese, Denti & Saija (2007).

Parameters

li	`int` Maximum field expansion order.
i	`int`
npnt	`int`
npntts	`int`
vk	`double` Wave number in scale units.
exdc	`double` External medium dielectric constant.
exri	`double` External medium refractive index.
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.
jer	`int &` Reference to integer error code variable.
lcalc	`int &` Reference to integer variable recording the maximum expansion order accounted for.
arg	`complex double &`
last_conf	`int` Last sphere configuration (used by CLUSTER)

9.33.2.6 envj()

```
double envj (
            int n,
            double x )
```

This function determines the control parameters required to calculate the Bessel functions up to the desired precision.

Parameters

n	`int`
x	`double`

Returns

result: `double`

9.33.2.7 mmulc()

```
void mmulc (
            dcomplex * vint,
```

```
            double ** cmullr,
            double ** cmul )
```

This function computes the Mueller Transformation Matrix, or Phase Matrix. See Sec. 2.8.1 of Borghese, Denti & Saija (2007).

Parameters

vint	`complex double *`
cmullr	`double **`
cmul	`double **`

9.33.2.8 msta1()

```
int msta1 (
            double x,
            int mp )
```

This function determines the starting point for backward recurrence such that the magnitude of all J and j functions is of the order of 10^{-mp}.

Parameters

x	`double` Absolute value of the argumetn to J or j.
mp	`int` Requested order of magnitude.

Returns

result: `int` The necessary starting point.

9.33.2.9 msta2()

```
int msta2 (
            double x,
            int n,
            int mp )
```

This function determines the starting point for backward recurrence such that all J and j functions have `mp` significant digits.

Parameters

x	`double` Absolute value of the argumetn to J or j.
n	`int` Order of the function.
mp	`int` Requested number of significant digits.

Returns

result: `int` The necessary starting point.

9.33.2.10 orunve()

```
void orunve (
            double * u1,
            double * u2,
            double * u3,
            int iorth,
            double torth )
```

This function computes the amplitude of the orthogonal unit vector for a geometry based either on the scattering plane or on the meridional plane. It is used by `upvsp()` and `upvmp()`. See Sec. 2.7 in Borghese, Denti & Saija (2007).

Parameters

u1	`double *`
u2	`double *`
u3	`double *`
iorth	`int`
torth	`double`

9.33.2.11 pwma()

```
void pwma (
            double * up,
            double * un,
            dcomplex * ylm,
            int inpol,
            int lw,
            int isq,
            ParticleDescriptor * c1 )
```

This function computes the amplitudes of the incident and scattered field on the basis of the multi-polar expansion. See Sec. 4.1.1 in Borghese, Denti and Saija (2007).

Parameters

up	`double *`
un	`double *`
ylm	`complex double *` Field polar spherical harmonics.
inpol	`int` Incident field polarization type (0 - linear; 1 - circular).
lw	`int`
isq	`int`
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.

9.33.2.12 rabas()

```
void rabas (
            int inpol,
            int li,
```

```
            int nsph,
            ParticleDescriptor * c1,
            double ** tqse,
            dcomplex ** tqspe,
            double ** tqss,
            dcomplex ** tqsps )
```

This function computes radiation torque on a sphere unit as the result of the difference between the extinction and the scattering contributions. See Sec. 4.9 in Borghese, Denti & Saija (2007).

Parameters

inpol	`int` Incident polarization type (0 - linear; 1 - circular)
li	`int` Maximum field expansion order.
nsph	`int` Number of spheres.
c1	`ParticleDescriptor *` Pointer to `ParticleDescriptor` data structure.
tqse	`double **`
tqspe	`complex double **`
tqss	`double **`
tqsps	`complex double **`

9.33.2.13 rbf()

```
void rbf (
            int n,
            double x,
            int & nm,
            double sj[] )
```

This function computes the real spherical Bessel funtions j. It uses the auxiliary functions `msta1()` and `msta2()` to determine the starting point for backward recurrence. This is the `SPHJ` implementation of the `specfun` library.

Parameters

n	`int` Order of the function.
x	`double` Argument of the function.
nm	`int &` Highest computed order.
sj	`double[]` The desired function j.

9.33.2.14 rkc()

```
void rkc (
            int npntmo,
            double step,
            dcomplex dcc,
            double & x,
            int lpo,
            dcomplex & y1,
            dcomplex & y2,
```

```
            dcomplex & dy1,
            dcomplex & dy2 )
```

This function determines the radial function and its derivative for a soft layer in a radially non homogeneous sphere. See Sec. 5.1 in Borghese, Denti & Saija (2007).

Parameters

npntmo	`int`
step	`double`
dcc	`complex double`
x	`double &`
lpo	`int`
y1	`complex double &`
y2	`complex double &`
dy1	`complex double &`
dy2	`complex double &`

9.33.2.15 rkt()

```
void rkt (
            int npntmo,
            double step,
            double & x,
            int lpo,
            dcomplex & y1,
            dcomplex & y2,
            dcomplex & dy1,
            dcomplex & dy2,
            ParticleDescriptor * c1 )
```

This function determines the radial function and its derivative for a transition layer in a radially non homogeneous sphere. See Sec. 5.1 in Borghese, Denti & Saija (2007).

Parameters

npntmo	`int`
step	`double`
x	`double &`
lpo	`int`
y1	`complex double &`
y2	`complex double &`
dy1	`complex double &`
dy2	`complex double &`
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.33.2.16 rnf()

```
void rnf (
            int n,
```

```
            double x,
            int & nm,
            double * sy )
```

This function computes the spherical Bessel functions y. It adopts the SPHJY implementation of the specfun library.

Parameters

n	int Order of the function (from 0 up).
x	double Argumento of the function ($x > 0$).
nm	int & Highest computed order.
sy	double * The desired function y.

9.33.2.17 sphar()

```
void sphar (
            double cosrth,
            double sinrth,
            double cosrph,
            double sinrph,
            int ll,
            dcomplex * ylm )
```

This function computes the field spherical harmonics for a given direction. See Sec. 1.5.2 in Borghese, Denti & Saija (2007).

Parameters

cosrth	double Cosine of direction's elevation.
sinrth	double Sine of direction's elevation.
cosrph	double Cosine of direction's azimuth.
sinrph	double Sine of direction's azimuth.
ll	int L value expansion order.
ylm	complex double * The requested spherical harmonics.

9.33.2.18 sscr0()

```
void sscr0 (
            dcomplex & tfsas,
            int nsph,
            int lm,
            double vk,
            double exri,
            ParticleDescriptor * c1 )
```

This function computes the scattering, absorption and extinction cross-sections in terms of Forward Scattering Amplitudes. See Sec. 4.2.1 in Borghese, Denti & Saija (2007).

Parameters

tfsas	`complex double &`
nsph	`int` Number of spheres.
lm	`int` Maximum field expansion order.
vk	`double` Wave number in scale units.
exri	`double` External medium refractive index.
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.

9.33.2.19 sscr2()

```
void sscr2 (
            int nsph,
            int lm,
            double vk,
            double exri,
            ParticleDescriptor * c1 )
```

The role of this function is to compute the scattering amplitude and the intensity of the scattered field. See Sec. 4.2 in Borghese, Denti & Saija (2007).

Parameters

nsph	`int` Number of spheres.
lm	`int` Maximum field expansion order.
vk	`double` Wave number in scale units.
exri	`double` External medium refractive index.
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.

9.33.2.20 thdps()

```
void thdps (
            int lm,
            double **** zpv )
```

This function computes the coefficients that enter the definition of the geometrical asymmetry parameter based on the L-value of the field expansion order. See Sec. 3.2.1 in Borghese, Denti & Saija (2007).

Parameters

lm	`int` Maximum field expansion order.
zpv	`double ****` Matrix of geometrical asymmetry parameter coefficients.

9.33.2.21 upvmp()

```
void upvmp (
            double thd,
```

```
            double phd,
            int icspnv,
            double & cost,
            double & sint,
            double & cosp,
            double & sinp,
            double * u,
            double * up,
            double * un )
```

This function computes the unitary vectors lying on the polarization plane and on its orthogonal direction, to optimize the identification of the scattering geometry. See Sec. 2.3 in Borghese, Denti & Saija (2007).

Parameters

thd	double
phd	double
icspnv	int
cost	double
sint	double
cosp	double
sinp	double
u	double *
up	double *
un	double *

9.33.2.22 upvsp()

```
void upvsp (
            double * u,
            double * upmp,
            double * unmp,
            double * us,
            double * upsmp,
            double * unsmp,
            double * up,
            double * un,
            double * ups,
            double * uns,
            double * duk,
            int & isq,
            int & ibf,
            double & scand,
            double & cfmp,
            double & sfmp,
            double & cfsp,
            double & sfsp )
```

This function computes the unitary vector perpendicular to the incident and scattering plane in a geometry based on the scattering plane. It uses orunve(). See Sec. 2.7 in Borghese, Denti & Saija (2007).

Parameters

u	double *

Parameters

upmp	`double *`
unmp	`double *`
us	`double *`
upsmp	`double *`
unsmp	`double *`
up	`double *`
un	`double *`
ups	`double *`
uns	`double *`
duk	`double *`
isq	`int &`
ibf	`int &`
scand	`double &` Scattering angle in degrees.
cfmp	`double &`
sfmp	`double &`
cfsp	`double &`
sfsp	`double &`

9.33.2.23 wmamp()

```
void wmamp (
            int iis,
            double cost,
            double sint,
            double cosp,
            double sinp,
            int inpol,
            int lm,
            int idot,
            int nsph,
            double * arg,
            double * u,
            double * up,
            double * un,
            ParticleDescriptor * c1 )
```

This function computes the coeffcients that define the geometrical symmetry parameter as defined with respect to the meridional plane. It makes use of `sphar()` and `pwma()`. See Sec. 3.2.1 in Borghese, Denti & Saija (2007).

Parameters

iis	`int`
cost	`double` Cosine of the elevation angle.
sint	`double` Sine of the elevation angle.
cosp	`double` Cosine of the azimuth angle.
sinp	`double` Sine of the azimuth angle.
inpol	`int` Incident field polarization type (0 - linear; 1 - circular).
lm	`int` Maximum field expansion orde.
idot	`int`

Parameters

nsph	`int` Number of spheres.
arg	`double *`
u	`double *`
up	`double *`
un	`double *`
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.

9.33.2.24 wmasp()

```
void wmasp (
            double cost,
            double sint,
            double cosp,
            double sinp,
            double costs,
            double sints,
            double cosps,
            double sinps,
            double * u,
            double * up,
            double * un,
            double * us,
            double * ups,
            double * uns,
            int isq,
            int ibf,
            int inpol,
            int lm,
            int idot,
            int nsph,
            double * argi,
            double * args,
            ParticleDescriptor * c1 )
```

This function computes the coefficients that define the geometrical asymmetry parameter based on the L-value with respect to the scattering plane. It uses `sphar()` and `pwma()`. See Sec. 3.2.1 in Borghese, Denti and Saija (2007).

Parameters

cost	`double` Cosine of elevation angle.
sint	`double` Sine of elevation angle.
cosp	`double` Cosine of azimuth angle.
sinp	`double` Sine of azimuth angle.
costs	`double` Cosine of scattering elevation angle.
sints	`double` Sine of scattering elevation angle.
cosps	`double` Cosine of scattering azimuth angle.
sinps	`double` Sine of scattering azimuth angle.
u	`double *`
up	`double *`
un	`double *`

Parameters

us	`double *`
ups	`double *`
uns	`double *`
isq	`int`
ibf	`int`
inpol	`int` Incident field polarization (0 - linear; 1 -circular).
lm	`int` Maximum field expansion order.
idot	`int`
nsph	`int` Number opf spheres.
argi	`double *`
args	`double *`
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.

9.34 sph_subs.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00030 #ifndef INCLUDE_SPH_SUBS_H_
00031 #define INCLUDE_SPH_SUBS_H_
00032
00045 void aps(double ****zpv, int li, int nsph, ParticleDescriptor *c1, double sqk, double *gaps);
00046
00058 void cbf(int n, dcomplex z, int &nm, dcomplex *csj);
00059
00071 double cg1(int lmpml, int mu, int l, int m);
00072
00086 void diel(int npntmo, int ns, int i, int ic, double vk, ParticleDescriptor *c1);
00087
00108 void dme(
00109      int li, int i, int npnt, int npntts, double vk, double exdc, double exri,
00110      ParticleDescriptor *c1, int &jer, int &lcalc, dcomplex &arg, int last_conf=0
00111 );
00112
00122 double envj(int n, double x);
00123
00133 void mmulc(dcomplex *vint, double **cmullr, double **cmul);
00134
00144 int msta1(double x, int mp);
00145
00156 int msta2(double x, int n, int mp);
00157
00170 void orunve(double *u1, double *u2, double *u3, int iorth, double torth);
00171
00185 void pwma(
00186       double *up, double *un, dcomplex *ylm, int inpol, int lw,
00187       int isq, ParticleDescriptor *c1
00188 );
00189
00205 void rabas(
00206        int inpol, int li, int nsph, ParticleDescriptor *c1, double **tqse, dcomplex **tqspe,
00207        double **tqss, dcomplex **tqsps
00208 );
00209
```

```
00221 void rbf(int n, double x, int &nm, double sj[]);
00222
00238 void rkc(
00239      int npntmo, double step, dcomplex dcc, double &x, int lpo,
00240      dcomplex &y1, dcomplex &y2, dcomplex &dy1, dcomplex &dy2
00241 );
00242
00258 void rkt(
00259      int npntmo, double step, double &x, int lpo, dcomplex &y1,
00260      dcomplex &y2, dcomplex &dy1, dcomplex &dy2, ParticleDescriptor *c1
00261 );
00262
00273 void rnf(int n, double x, int &nm, double *sy);
00274
00287 void sphar(
00288        double cosrth, double sinrth, double cosrph, double sinrph,
00289        int ll, dcomplex *ylm
00290 );
00291
00304 void sscr0(dcomplex &tfsas, int nsph, int lm, double vk, double exri, ParticleDescriptor *c1);
00305
00317 void sscr2(int nsph, int lm, double vk, double exri, ParticleDescriptor *c1);
00318
00328 void thdps(int lm, double ****zpv);
00329
00348 void upvmp(
00349        double thd, double phd, int icspnv, double &cost, double &sint,
00350        double &cosp, double &sinp, double *u, double *up, double *un
00351 );
00352
00378 void upvsp(
00379        double *u, double *upmp, double *unmp, double *us, double *upsmp, double *unsmp,
00380        double *up, double *un, double *ups, double *uns, double *duk, int &isq,
00381        int &ibf, double &scand, double &cfmp, double &sfmp, double &cfsp, double &sfsp
00382 );
00383
00405 void wmamp(
00406        int iis, double cost, double sint, double cosp, double sinp, int inpol,
00407        int lm, int idot, int nsph, double *arg, double *u, double *up,
00408        double *un, ParticleDescriptor *c1
00409 );
00410
00441 void wmasp(
00442        double cost, double sint, double cosp, double sinp, double costs, double sints,
00443        double cosps, double sinps, double *u, double *up, double *un, double *us,
00444        double *ups, double *uns, int isq, int ibf, int inpol, int lm, int idot,
00445        int nsph, double *argi, double *args, ParticleDescriptor *c1
00446 );
00447
00448 #endif /* SRC_INCLUDE_SPH_SUBS_H_ */
```

9.35 np_tmcode/src/include/tfrfme.h File Reference

Representation of the trapping calculation objects.

Classes

- class Swap1

  Class to represent the first group of trapping swap data.
- class Swap2

  Class to represent the second group of trapping swap data.
- class TFRFME

  Class to represent the trapping configuration.

9.36 tfrfme.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
```

```
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00022 #ifndef INCLUDE_TFRFME_H_
00023 #define INCLUDE_TFRFME_H_
00024
00027 class Swap1 {
00028 protected:
00030   int _last_index;
00032   int _nkv;
00034   int _nlmmt;
00035
00037   dcomplex *_wk;
00038
00044   static Swap1 *from_hdf5(const std::string& file_name);
00045
00051   static Swap1 *from_legacy(const std::string& file_name);
00052
00057   void write_hdf5(const std::string& file_name);
00058
00063   void write_legacy(const std::string& file_name);
00064
00065 public:
00067   const dcomplex *wk;
00068
00074   Swap1(int lm, int nkv);
00075
00078   ~Swap1() { delete[] _wk; }
00079
00084   void append(dcomplex value) { _wk[_last_index++] = value; }
00085
00093   static Swap1* from_binary(const std::string& file_name, const std::string& mode="LEGACY");
00094
00101   static long get_size(int lm, int nkv);
00102
00105   void reset() { _last_index = 0; }
00106
00113   void write_binary(const std::string& file_name, const std::string& mode="LEGACY");
00114
00122   bool operator ==(Swap1 &other);
00123 };
00124
00127 class Swap2 {
00128 protected:
00130   int _last_vector;
00132   int _last_matrix;
00134   int _nkv;
00136   double _apfafa;
00138   double _pmf;
00140   double _spd;
00142   double _rir;
00144   double _ftcn;
00146   double _fshmx;
00148   double _vxyzmx;
00150   double _delxyz;
00152   double _vknmx;
00154   double _delk;
00156   double _delks;
00158   int _nlmmt;
00160   int _nrvc;
00161
00167   static Swap2 *from_hdf5(const std::string& file_name);
00168
00174   static Swap2 *from_legacy(const std::string& file_name);
00175
00180   void write_hdf5(const std::string& file_name);
00181
00186   void write_legacy(const std::string& file_name);
00187
00188 public:
00190   const int &last_vector = _last_vector;
00192   const int &last_matrix = _last_matrix;
00194   const int &nkv = _nkv;
00196   double *vkv;
00198   double *vec_vkzm;
```

```
00200   const double &apfafa = _apfafa;
00202   const double &pmf = _pmf;
00204   const double &spd = _spd;
00206   const double &rir = _rir;
00208   const double &ftcn = _ftcn;
00210   const double &fshmx = _fshmx;
00212   const double &vxyzmx = _vxyzmx;
00214   const double &delxyz = _delxyz;
00216   const double &vknmx = _vknmx;
00218   const double &delk = _delk;
00220   const double &delks = _delks;
00222   const int &nlmmt = _nlmmt;
00224   const int &nrvc = _nrvc;
00225
00230   Swap2(int nkv);
00231
00234   ~Swap2();
00235
00243   static Swap2* from_binary(const std::string& file_name, const std::string& mode="LEGACY");
00244
00250   static long get_size(int nkv);
00251
00256   double *get_vector() { return vkv; }
00257
00262   void push_matrix(double value);
00263
00268   void push_vector(double value) { vkv[_last_vector++] = value; }
00269
00272   void reset_matrix() { _last_matrix = 0; }
00273
00276   void reset_vector() { _last_vector = 0; }
00277
00283   void set_param(const std::string& param_name, double value);
00284
00291   void write_binary(const std::string& file_name, const std::string& mode="LEGACY");
00292
00300   bool operator ==(Swap2 &other);
00301 };
00302
00305 class TFRFME {
00306 protected:
00308   int _nlmmt;
00310   int _nrvc;
00312   int _lmode;
00314   int _lm;
00316   int _nkv;
00318   int _nxv;
00320   int _nyv;
00322   int _nzv;
00324   double _vk;
00326   double _exri;
00328   double _an;
00330   double _ff;
00332   double _tra;
00334   double _spd;
00336   double _frsh;
00338   double _exril;
00340   double *xv;
00342   double *yv;
00344   double *zv;
00345
00352   static TFRFME *from_hdf5(const std::string& file_name);
00353
00360   static TFRFME *from_legacy(const std::string& file_name);
00361
00366   void write_hdf5(const std::string& file_name);
00367
00372   void write_legacy(const std::string& file_name);
00373
00374 public:
00376   const int& nlmmt = _nlmmt;
00378   const int& nrvc = _nrvc;
00380   const int& lmode = _lmode;
00382   const int& lm = _lm;
00384   const int& nkv = _nkv;
00386   const int& nxv = _nxv;
00388   const int& nyv = _nyv;
00390   const int& nzv = _nzv;
00392   const double& vk = _vk;
00394   const double& exri = _exri;
00396   const double& an = _an;
00398   const double& ff = _ff;
00400   const double& tra = _tra;
00402   const double& spd = _spd;
00404   const double& frsh = _frsh;
00406   const double& exril = _exril;
00408   dcomplex *vec_wsum;
```

```
00409
00419   TFRFME(int lmode, int lm, int nkv, int nxv, int nyv, int nzv);
00420
00423   ~TFRFME();
00424
00433   static TFRFME* from_binary(
00434     const std::string& file_name, const std::string& mode="LEGACY"
00435 );
00436
00446   static long get_size(int lm, int nkv, int nxv, int nyv, int nzv);
00447
00452   double *get_x() { return xv; }
00453
00458   double *get_y() { return yv; }
00459
00464   double *get_z() { return zv; }
00465
00471   void set_param(const std::string& param_name, double value);
00472
00479   void write_binary(const std::string& file_name, const std::string& mode="LEGACY");
00480
00488   bool operator ==(const TFRFME& other);
00489 };
00490 #endif
```

9.37 np_tmcode/src/include/tra_subs.h File Reference

C++ porting of TRAPPING functions and subroutines.

Classes

- struct CIL

  CIL data structure.
- struct CCR

  CCR data structure.

Functions

- void camp (dcomplex *ac, dcomplex **am0m, dcomplex *ws, CIL *cil)
- void czamp (dcomplex *ac, dcomplex **amd, int **indam, dcomplex *ws, CIL *cil)
- void ffrf (double ****zpv, dcomplex *ac, dcomplex *ws, double *fffe, double *fffs, CIL *cil, CCR *ccr)
- void ffrt (dcomplex *ac, dcomplex *ws, double *ffte, double *ffts, CIL *cil)
- dcomplex * frfmer (int nkv, double vkm, double vknmx, double apfafa, double tra, double spd, double rir, double ftcn, int le, int lmode, double pmf, Swap1 *tt1, Swap2 *tt2)
- void pwmalp (dcomplex **w, double *up, double *un, dcomplex *ylm, int lw)
- void samp (dcomplex *ac, dcomplex *tmsm, dcomplex *tmse, dcomplex *ws, CIL *cil)
- void sampoa (dcomplex *ac, dcomplex **tms, dcomplex *ws, CIL *cil)
- void wamff (dcomplex *wk, double x, double y, double z, int lm, double apfafa, double tra, double spd, double rir, double ftcn, int lmode, double pmf)

9.37.1 Detailed Description

This library includes a collection of functions that are used to solve the trapping problem. The functions that were generalized from the case of the single sphere are imported the `sph_subs.h` library. As it occurs with the single sphere case functions, in most cases, the results of calculations do not fall back to fundamental data types. They are rather multi-component structures. In order to manage access to such variety of return values, most functions are declared as `void` and they operate on output arguments passed by reference.

9.37.2 Function Documentation

9.37.2.1 camp()

```
void camp (
            dcomplex * ac,
            dcomplex ** am0m,
            dcomplex * ws,
            CIL * cil )
```

C++ porting of CAMP

Parameters

ac	`complex double *` ANNOTATION: vector of complex amplitudes.
am0m	`complex double **` ANNOTATION: T-matrix of the particle.
ws	`complex double *` ANNOTATION: scattering amplitudes.
cil	`CIL *` Pointer to a CIL structure.

This function builds the AC vector using AM0M and WS.

9.37.2.2 czamp()

```
void czamp (
            dcomplex * ac,
            dcomplex ** amd,
            int ** indam,
            dcomplex * ws,
            CIL * cil )
```

C++ porting of CZAMP

Parameters

ac	Vector of complex. ANNOTATION: vector of complex amplitudes.
amd	Matrix of complex. ANNOTATION: T-matrix.
indam	`int **`. ANNOTATION: map of matrix indexes.
ws	Vector of complex. ANNOTATION: scattering amplitudes.
cil	`CIL *` Pointer to a CIL structure.

This function builds the AC vector using AMD, INDAM and WS.

9.37.2.3 ffrf()

```
void ffrf (
            double **** zpv,
            dcomplex * ac,
            dcomplex * ws,
            double * fffe,
```

```
            double * fffs,
            CIL * cil,
            CCR * ccr )
```

C++ porting of FFRF

Parameters

zpv	`double ****`. ANNOTATION: asymmetry tensor.
ac	`complex double *` ANNOTATION: vector of complex amplitudes.
ws	`complex double *` ANNOTATION: scattering amplitudes.
fffe	`double *`. ANNOTATION: extinction contributions to force.
fffs	`double *`. ANNOTATION: scattering contributions to force.
cil	`CIL *` Pointer to a CIL structure.
ccr	`CCR *` Pointer to a CCR structure.

9.37.2.4 ffrt()

```
void ffrt (
            dcomplex * ac,
            dcomplex * ws,
            double * ffte,
            double * ffts,
            CIL * cil )
```

C++ porting of FFRT

Parameters

ac	`complex double *` ANNOTATION: vector of complex amplitudes.
ws	`complex double *` ANNOTATION: scattering amplitudes.
ffte	`double *`. ANNOTATION: extinction contributions to torque.
ffts	`double *`. ANNOTATION: scattering contributions to torque.
cil	`CIL *` Pointer to a CIL structure.

9.37.2.5 frfmer()

```
dcomplex * frfmer (
            int nkv,
            double vkm,
            double vknmx,
            double apfafa,
            double tra,
            double spd,
            double rir,
            double ftcn,
            int le,
            int lmode,
            double pmf,
```

```
            Swap1 * tt1,
            Swap2 * tt2 )
```

C++ porting of FRFMER

Parameters

nkv	`int` ANNOTATION: number of wave vectors.
vkm	`double` ANNOTATION: wave number.
vknmx	`double` ANNOTATION: maximum wave number.
apfafa	`double` ANNOTATION: normalized aperture.
tra	`double` ANNOTATION: lens transmission.
spd	`double` ANNOTATION: cover thickness.
rir	`double` ANNOTATION: external over oil refractive index ratio.
ftcn	`double` ANNOTATION: refraction factor.
le	`int` ANNOTATION: external multipole truncation order.
lmode	`int` ANNOTATION: laser mode.
pmf	`double` ANNOTATION: aperture correction.
tt1	`Swap1 *` Pointer to first swap object.
tt2	`Swap2 *` Pointer to second swap object.

Returns

wk: `complex double *` ANNOTATION: incident amplitudes.

9.37.2.6 pwmalp()

```
void pwmalp (
            dcomplex ** w,
            double * up,
            double * un,
            dcomplex * ylm,
            int lw )
```

C++ porting of PWMALP

Parameters

w	`complex double *` ANNOTATION: amplitudes.
up	`double *` ANNOTATION: unit polarization vector.
un	`double *` ANNOTATION: unit normal vector.
ylm	`complex double *` Field vector spherical harmonics.
lw	`int` ANNOTATION: order of amplitudes.

9.37.2.7 samp()

```
void samp (
            dcomplex * ac,
```

```
            dcomplex * tmsm,
            dcomplex * tmse,
            dcomplex * ws,
            CIL * cil )
```

C++ porting of SAMP

Parameters

ac	`complex double *` ANNOTATION: vector of complex amplitudes.
tmsm	`complex double *` ANNOTATION: T-matrix M components.
tmse	`complex double *` ANNOTATION: T-matrix E components.
ws	`complex double *` ANNOTATION: scattering amplitudes.
cil	`CIL *` Pointer to a CIL structure.

This function builds the AC vector using TMSM, TMSE and WS.

9.37.2.8 sampoa()

```
void sampoa (
            dcomplex * ac,
            dcomplex ** tms,
            dcomplex * ws,
            CIL * cil )
```

C++ porting of SAMPOA

Parameters

ac	`complex double *` ANNOTATION: vector of complex amplitudes.
tms	`complex double **` ANNOTATION: T-matrix.
ws	`complex double *` ANNOTATION: scattering amplitudes.
cil	`CIL *` Pointer to a CIL structure.

This function builds the AC vector using TMS and WS.

9.37.2.9 wamff()

```
void wamff (
            dcomplex * wk,
            double x,
            double y,
            double z,
            int lm,
            double apfafa,
            double tra,
            double spd,
            double rir,
            double ftcn,
            int lmode,
            double pmf )
```

C++ porting of WAMFF

Parameters

wk	`complex double *` ANNOTATION: incident amplitudes.
x	`double` ANNOTATION: X coordinate.
y	`double` ANNOTATION: Y coordinate.
z	`double` ANNOTATION: Z coordinate.
lm	`int` ANNOTATION: maximum multipole truncation order.
apfafa	`double` ANNOTATION: Normalized aperture.
tra	`double` ANNOTATION: Lens transmission.
spd	`double` ANNOTATION: Cover slip thickness.
rir	`double` ANNOTATION: External over oil refractive index ratio.
ftcn	`double` ANNOTATION: Refraction factor.
lmode	`int` ANNOTATION: Laser mode.
pmf	`double` ANNOTATION: aperture correction.

9.38 tra_subs.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00031 #ifndef INCLUDE_TRA_SUBS_H_
00032 #define INCLUDE_TRA_SUBS_H_
00033 #endif
00034
00035 // Structures for TRAPPING
00040 struct CIL {
00042   int le;
00044   int nlem;
00046   int nlemt;
00048   int mxmpo;
00050   int mxim;
00051 };
00052
00057 struct CCR {
00059   double cof;
00061   double cimu;
00062 };
00063 //End of TRAPPING structures
00064
00074 void camp(dcomplex *ac, dcomplex **am0m, dcomplex *ws, CIL *cil);
00075
00086 void czamp(dcomplex *ac, dcomplex **amd, int **indam, dcomplex *ws, CIL *cil);
00087
00098 void ffrf(
00099       double ****zpv, dcomplex *ac, dcomplex *ws, double *fffe,
00100       double *fffs, CIL *cil, CCR *ccr
00101 );
00102
00111 void ffrt(dcomplex *ac, dcomplex *ws, double *ffte, double *ffts, CIL *cil);
00112
00130 dcomplex *frfmer(
00131         int nkv, double vkm, double vknmx, double apfafa, double tra,
00132         double spd, double rir, double ftcn, int le, int lmode, double pmf,
00133         Swap1 *tt1, Swap2 *tt2
00134 );
00135
```

```
00144 void pwmalp(dcomplex **w, double *up, double *un, dcomplex *ylm, int lw);
00145
00156 void samp(dcomplex *ac, dcomplex *tmsm, dcomplex *tmse, dcomplex *ws, CIL *cil);
00157
00167 void sampoa(dcomplex *ac, dcomplex **tms, dcomplex *ws, CIL *cil);
00168
00184 void wamff(
00185        dcomplex *wk, double x, double y, double z, int lm, double apfafa,
00186        double tra, double spd, double rir, double ftcn, int lmode, double pmf
00187 );
```

9.39 np_tmcode/src/include/TransitionMatrix.h File Reference

Representation of the Transition Matrix.

Classes

- class TransitionMatrix

  Class to represent the Transition Matrix.

9.40 TransitionMatrix.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00022 #ifndef INCLUDE_TRANSITIONMATRIX_H_
00023 #define INCLUDE_TRANSITIONMATRIX_H_
00024
00027 class TransitionMatrix {
00028  protected:
00030   int _is;
00032   int _l_max;
00034   double _vk;
00036   double _exri;
00038   double _sphere_radius;
00039
00049   static TransitionMatrix *from_hdf5(const std::string& file_name);
00050
00057   static TransitionMatrix *from_legacy(const std::string& file_name);
00058
00067   void write_hdf5(const std::string& file_name);
00068
00083   static void write_hdf5(
00084     const std::string& file_name, np_int nlemt, int lm, double vk,
00085     double exri, dcomplex **am0m
00086   );
00087
00103   static void write_hdf5(
00104     const std::string& file_name, int lm, double vk, double exri,
00105     dcomplex **rmi, dcomplex **rei, double sphere_radius
00106   );
00107
00112   void write_legacy(const std::string& file_name);
00113
00128   static void write_legacy(
00129     const std::string& file_name, np_int nlemt, int lm, double vk,
```

```
00130     double exri, dcomplex **am0m
00131   );
00132
00148   static void write_legacy(
00149     const std::string& file_name, int lm, double vk, double exri,
00150     dcomplex **rmi, dcomplex **rei, double radius
00151   );
00152
00153  public:
00155   const int& is = _is;
00157   const int& l_max = _l_max;
00159   const double& vk = _vk;
00161   const double& exri = _exri;
00163   dcomplex *elements;
00165   const double& sphere_radius = _sphere_radius;
00167   int *shape;
00168
00178   TransitionMatrix(
00179     int is, int lm, double vk, double exri, dcomplex *_elems, double radius=0.0
00180   );
00181
00194   TransitionMatrix(
00195      int lm, double vk, double exri, dcomplex **rmi, dcomplex **rei, double radius
00196   );
00197
00209   TransitionMatrix(int nlemt, int lm, double vk, double exri, dcomplex **am0m);
00210
00213   ~TransitionMatrix();
00214
00231   static TransitionMatrix* from_binary(const std::string& file_name, const std::string&
      mode="LEGACY");
00232
00245   void write_binary(const std::string& file_name, const std::string& mode="LEGACY");
00246
00269   static void write_binary(
00270      const std::string& file_name, np_int nlemt, int lm, double vk,
00271      double exri, dcomplex **am0m, const std::string& mode="LEGACY"
00272   );
00273
00297   static void write_binary(
00298     const std::string& file_name, int lm, double vk, double exri,
00299     dcomplex **rmi, dcomplex **rei, double sphere_radius,
00300     const std::string& mode="LEGACY"
00301   );
00302
00312   bool operator ==(TransitionMatrix &other);
00313 };
00314
00315 #endif
```

9.41 np_tmcode/src/include/types.h File Reference

Definition of fundamental types in use.

```
#include <complex.h>
```

Macros

- #define **np_int** int32_t

  Alias of the default integer type.
- #define **dconjg**(z) ((__real__ (z) - I * (__imag__ (z))))

  Macro to compute the conjugate of a complex number.

Typedefs

- typedef __complex__ double **dcomplex**

  Short-cut to C-style double precision complex type.

Functions

- double imag (const dcomplex &z)

  Get the imaginary part of double precision complex number.
- double real (const dcomplex &z)

  Get the imaginary part of double precision complex number.

9.41.1 Function Documentation

9.41.1.1 imag()

```
double imag (
            const dcomplex & z )  [inline]
```

Parameters

z	`const dcomplex &` Reference to the complex number from which to extract the imaginary part.

Returns

Im(z): `double` Imaginary part of z.

9.41.1.2 real()

```
double real (
            const dcomplex & z )  [inline]
```

Parameters

z	`const dcomplex &` Reference to the complex number from which to extract the real part.

Returns

Re(z): `double` Real part of z.

9.42 types.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
```

```
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00022 #ifndef INCLUDE_TYPES_H_
00023 #define INCLUDE_TYPES_H_
00024
00025 #include <complex.h>
00026
00028 typedef __complex__ double dcomplex;
00029
00030 #ifdef USE_MKL
00031 #ifdef USE_ILP64
00032 #ifndef MKL_INT
00033 #define MKL_INT int64_t
00034 #endif // MKL_INT
00035 #else
00036 #ifndef MKL_INT
00037 #define MKL_INT int32_t
00038 #endif // MKL_INT
00039 #endif // USE_ILP64
00040 #endif // USE_MKL
00041
00042 #ifdef USE_LAPACK
00043 #ifdef USE_MKL
00044 #include <mkl_lapacke.h>
00045 #else
00046 #include <lapacke.h>
00047 #endif // USE_MKL
00048 #endif // USE_LAPACK
00049
00050 #ifdef USE_MAGMA
00051 #include <magma_v2.h>
00052 #endif
00053
00054 #ifndef np_int
00055 #ifdef lapack_int
00057 #define np_int lapack_int
00058 #else
00059 #ifdef USE_ILP64
00061 #define np_int int64_t
00062 #else
00064 #define np_int int32_t
00065 #endif // USE_ILP64
00066 #endif // lapack_int
00067 #endif // np_int
00068
00070 #define dconjg(z) ( (__real__ (z) - I * (__imag__ (z))) )
00071
00078 double inline imag(const dcomplex &z) { return __imag__ z; }
00079
00086 double inline real(const dcomplex &z) { return __real__ z; }
00087 #endif
```

9.43 np_tmcode/src/include/utils.h File Reference

Definition of auxiliary code utilities.

Functions

- double get_ram_overhead ()

  Obtain an estimate of the RAM overhead factor for optimization.
- int write_dcomplex_matrix (VirtualAsciiFile *af, dcomplex **mat, int rows, int columns, const std::string &format=" %5d %5d (%17.8lE,%17.8lE)\n", int first_index=1)

  Write a double complex matrix to a text file.

9.43.1 Function Documentation

9.43.1.1 get_ram_overhead()

```
double get_ram_overhead ( )
```

Code speed-up optimization usually comes at the cost of increased memory requirements. While this should not generally be a serious issue, it can become such in case of models that require large amounts of memory to be handled. This function tests the code build configuration to provide an zero-order estimate of the weight of these overheads to protect the host system from saturating the RAM.

Returns

factor: `double` The multiplicative factor to be applied to data size.

9.43.1.2 write_dcomplex_matrix()

```
int write_dcomplex_matrix (
              VirtualAsciiFile * af,
              dcomplex ** mat,
              int rows,
              int columns,
              const std::string & format = " %5d %5d (%17.8lE,%17.8lE)\n",
              int first_index = 1 )
```

Parameters

af	`VirtualAsciiFile *` Pointer to an existing VirtualAsciiFile.
mat	`dcomplex **` Pointer to the matrix.
rows	`int` Number of rows in the matrix.
columns	`int` Number of columns in the matrix.
format	`const string&` Format of the line (default is " %5d %5d (%17.8lE,%17.8lE) ")
first_index	`int` Index of the first element (default is 1, i.e. base 1 FORTRAN array notation)

9.44 utils.h

Go to the documentation of this file.

```
00001 /* Copyright (C) 2025   INAF - Osservatorio Astronomico di Cagliari
00002
00003    This program is free software: you can redistribute it and/or modify
00004    it under the terms of the GNU General Public License as published by
00005    the Free Software Foundation, either version 3 of the License, or
00006    (at your option) any later version.
00007
00008    This program is distributed in the hope that it will be useful,
00009    but WITHOUT ANY WARRANTY; without even the implied warranty of
00010    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00011    GNU General Public License for more details.
00012
00013    A copy of the GNU General Public License is distributed along with
00014    this program in the COPYING file. If not, see: <https://www.gnu.org/licenses/>.
00015  */
00016
00022 #ifndef INCLUDE_UTILS_H_
00023 #define INCLUDE_UTILS_H_
00024
00036 double get_ram_overhead();
00037
00047 int write_dcomplex_matrix(
00048               VirtualAsciiFile *af, dcomplex **mat, int rows,
00049               int columns, const std::string& format=" %5d %5d (%17.8lE,%17.8lE)\n",
00050               int first_index=1
00051 );
00052
00053 #endif
```

9.45 np_tmcode/src/inclusion/inclusion.cpp File Reference

Implementation of the calculation for a sphere with inclusions.

```
#include <chrono>
#include <cmath>
#include <cstdio>
#include <exception>
#include <fstream>
#include <hdf5.h>
#include <string>
#include "../include/types.h"
#include "../include/errors.h"
#include "../include/logging.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
#include "../include/sph_subs.h"
#include "../include/clu_subs.h"
#include "../include/inclu_subs.h"
#include "../include/TransitionMatrix.h"
#include "../include/algebraic.h"
#include "../include/List.h"
#include "../include/file_io.h"
#include "../include/utils.h"
#include "../include/outputs.h"
#include "../include/IterationData.h"
```

Functions

- int inclusion_jxi488_cycle (int jxi488, ScattererConfiguration *sconf, GeometryConfiguration *gconf, ScatteringAngles *sa, InclusionIterationData *cid, InclusionOutputInfo *output, const string &output_path, VirtualBinaryFile *vtppoanp)

  Main calculation loop.
- void inclusion (const string &config_file, const string &data_file, const string &output_path, const mixMPI *mpidata)

  C++ implementation of INCLU.

9.45.1 Function Documentation

9.45.1.1 inclusion()

```
void inclusion (
            const string & config_file,
            const string & data_file,
            const string & output_path,
            const mixMPI * mpidata )
```

Parameters

config_file	`string` Name of the configuration file.
data_file	`string` Name of the input data file.
output_path	`string` Directory to write the output files in.
mpidata	`mixMPI *` Pointer to an instance of MPI data settings.

9.45.1.2 inclusion_jxi488_cycle()

```
int inclusion_jxi488_cycle (
            int jxi488,
            ScattererConfiguration * sconf,
            GeometryConfiguration * gconf,
            ScatteringAngles * sa,
            InclusionIterationData * cid,
            InclusionOutputInfo * output,
            const string & output_path,
            VirtualBinaryFile * vtppoanp )
```

The solution of the scattering problem for different wavelengths is an embarrasingly parallel task. This function, therefore, collects all the operations that can be independently executed by different processes, after the configuration stage and the first calculation loop have been executed.

Parameters

jxi488	`int` Wavelength loop index.
sconf	`ScattererConfiguration *` Pointer to a `ScattererConfiguration` object.
gconf	`GeometryConfiguration *` Pointer to a `GeometryConfiguration` object.
sa	`ScatteringAngles *` Pointer to a `ScatteringAngles` object.
cid	`InclusionIterationData *` Pointer to an `InclusionIterationData` object.
output	`InclusionOutputInfo *` Pointer to an `InclusionOutputInfo` object.
output_path	`const string &` Path to the output directory.
vtppoanp	`VirtualBinaryFile *` Pointer to a `VirtualBinaryFile` object.

9.46 np_tmcode/src/inclusion/np_inclusion.cpp File Reference

Sphere with inclusions scattering program handler.

```
#include <cstdio>
#include <string>
#include "../include/types.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
```

Functions

- void inclusion (const string &config_file, const string &data_file, const string &output_path, const mixMPI *mpidata)

  C++ implementation of INCLU.
- int main (int argc, char **argv)

  Main program entry point.

9.46.1 Detailed Description

This program emulates the execution work-flow originally handled by the FORTRAN EDFB_INCLU and INCLU codes, which undertook the task of solving the scattering calculation for the case of a sphere with inclusions. The program is designed to work in two modes: emulation and enhanced. The emulation mode is activated by invoking the program without arguments. In this case, the code looks for hard-coded default input and writes output in the execution folder, replicating the behaviour of the original FORTRAN code. Enhanced mode, instead, is activated by passing command line arguments that locate the desired input files and a valid folder to write the output into. The emulation mode is useful for testing, while the advanced mode implements the possibility to change input and output options, without having to modify the code.

9.46.2 Function Documentation

9.46.2.1 inclusion()

```
void inclusion (
            const string & config_file,
            const string & data_file,
            const string & output_path,
            const mixMPI * mpidata )  [extern]
```

Parameters

config_file	`string` Name of the configuration file.
data_file	`string` Name of the input data file.
output_path	`string` Directory to write the output files in.
mpidata	`mixMPI *` Pointer to an instance of MPI data settings.

9.46.2.2 main()

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

This is the starting point of the execution flow. Here we may choose how to configure the code, e.g. by loading a legacy configuration file or some otherwise formatted configuration data set. The code can be linked to a luncher script or to a GUI oriented application that performs the configuration and runs the main program.

Parameters

argc	`int` The number of arguments given in command-line.
argv	`char **` The vector of command-line arguments.

Returns

result: `int` An exit code passed to the OS (0 for succesful execution).

9.47 np_tmcode/src/libnptm/algebraic.cpp File Reference

Implementation of algebraic functions with different call-backs.

```
#include <string>
#include "../include/types.h"
#include "../include/algebraic.h"
```

Functions

- void lucin (dcomplex **am, const np_int nddmst, np_int n, int &ier)

  Invert the multi-centered M-matrix.
- void invert_matrix (dcomplex **mat, np_int size, int &ier, int &maxrefiters, double &accuracygoal, int refinemode, const string &output_path, int jxi488, np_int max_size, int target_device)

  Perform in-place matrix inversion.

9.47.1 Function Documentation

9.47.1.1 invert_matrix()

```
void invert_matrix (
            dcomplex ** mat,
            np_int size,
            int & ier,
            int & maxrefiters,
            double & accuracygoal,
            int refinemode,
            const string & output_path,
            int jxi488,
            np_int max_size = 0,
            int target_device = 0 )
```

Parameters

mat	`complex double **` The matrix to be inverted (must be a square matrix).
size	`np_int` The size of the matrix (i.e. the number of its rows or columns).
ier	`int &` Reference to an integer variable for returning a result flag.
maxrefiters	`int &` Reference to the maximum number of refinement iterations.
accuracygoal	`double &` Reference to the requested accuracy level.
refinemode	`int` Flag for refinement mode selection.
output_path	`const string &` Path where the output needs to be placed.
jxi488	`int` Index of the current wavelength calculation.
max_size	`np_int` The maximum expected size (required by some call-backs, optional, defaults to 0).
target_device	`int` ID of target GPU, if available (defaults to 0).

9.47.1.2 lucin()

```
void lucin (
            dcomplex ** am,
            const np_int nddmst,
            np_int n,
            int & ier )  [extern]
```

This function performs the inversion of the multi-centered M-matrix through LU decomposition. See Eq. (5.29) in Borghese, Denti & Saija (2007).

Parameters

am	complex double **
nddmst	const int64_t
n	int64_t
ier	int &

9.48 np_tmcode/src/libnptm/clu_subs.cpp File Reference

C++ implementation of CLUSTER subroutines.

```
#include "../include/types.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
#include "../include/sph_subs.h"
#include "../include/clu_subs.h"
```

Functions

- void apc (double ****zpv, int le, dcomplex **am0m, dcomplex **w, double sqk, double **gapr, dcomplex **gapp)

  Compute the asymmetry-corrected scattering cross-section of a cluster.
- void apcra (double ****zpv, const int le, dcomplex **am0m, int inpol, double sqk, double **gaprm, dcomplex **gappm)

  Compute the asymmetry-corrected scattering cross-section under random average conditions.
- dcomplex cdtp (dcomplex z, dcomplex *vec_am, int i, int jf, int k, int nj, np_int istep)

  Complex inner product.
- double cgev (int ipamo, int mu, int l, int m)

  C++ porting of CGEV. ANNOTATION: Get weight of T-matrix element.
- void cms (dcomplex **am, ParticleDescriptor *c1)

  Build the multi-centered M-matrix of the cluster.
- void crsm1 (double vk, double exri, ParticleDescriptor *c1)

  Compute orientation-averaged scattered field intensity.
- dcomplex ghit_d (int ihi, int ipamo, int nbl, int l1, int m1, int l2, int m2, ParticleDescriptor *c1, double *rac3j)

  Compute the transfer vector from N2 to N1.
- dcomplex ghit (int ihi, int ipamo, int nbl, int l1, int m1, int l2, int m2, ParticleDescriptor *c1)

  Compute the transfer vector from N2 to N1.
- void hjv (double exri, double vk, int &jer, int &lcalc, dcomplex &arg, ParticleDescriptor *c1)

Compute Hankel funtion and Bessel functions.

- void lucin (dcomplex **am, const np_int nddmst, np_int n, int &ier)

  Invert the multi-centered M-matrix.
- void mextc (double vk, double exri, dcomplex **fsac, double **cextlr, double **cext)

  Compute the average extinction cross-section.
- void pcros (double vk, double exri, ParticleDescriptor *c1)

  Compute cross-sections and forward scattering amplitude for the cluster.
- void pcrsm0 (double vk, double exri, int inpol, ParticleDescriptor *c1)

  Compute orientation-averaged cross-sections and forward scattering amplitude.
- void polar (double x, double y, double z, double &r, double &cth, double &sth, double &cph, double &sph)

  Transform Cartesian quantities to spherical coordinates ones.
- void r3j000 (int j2, int j3, double *rac3j)

  Compute the 3j symbol for Clebsch-Gordan coefficients towards J=0.
- void r3jjr (int j2, int j3, int m2, int m3, double *rac3j)

  Compute the 3j symbol for Clebsch-Gordan coefficients for JJ transitions.
- void r3jjr_d (int j2, int j3, int m2, int m3, double *rac3j)

  Compute the 3j symbol for Clebsch-Gordan coefficients for JJ transitions.
- void r3jmr (int j1, int j2, int j3, int m1, double *rac3j)

  Compute the 3j symbol for Clebsch-Gordan coefficients for JM transitions.
- void raba (int le, dcomplex **am0m, dcomplex **w, double **tqce, dcomplex **tqcpe, double **tqcs, dcomplex **tqcps)

  Compute radiation torques on a particle in Cartesian coordinates.
- void rftr (double *u, double *up, double *un, double *gapv, double extins, double scatts, double &rapr, double &cosav, double &fp, double &fn, double &fk, double &fx, double &fy, double &fz)

  Compute the radiation force Cartesian components.
- void scr0 (double vk, double exri, ParticleDescriptor *c1)

  Compute Mie cross-sections for the sphere units in the cluster.
- void scr2 (double vk, double vkarg, double exri, double *duk, ParticleDescriptor *c1)

  Compute the scattering amplitude for a single sphere in an aggregate.
- void str (double **rcf, ParticleDescriptor *c1)

  Transform sphere Cartesian coordinates to spherical coordinates.
- void tqr (double *u, double *up, double *un, double *tqev, double *tqsv, double &tep, double &ten, double &tek, double &tsp, double &tsn, double &tsk)

  Compute radiation torques on particles on a k-vector oriented system.
- void ztm (dcomplex **am, ParticleDescriptor *c1)

  Calculate the single-centered inversion of the M-matrix.

9.48.1 Function Documentation

9.48.1.1 apc()

```
void apc (
            double **** zpv,
            int le,
            dcomplex ** am0m,
            dcomplex ** w,
            double sqk,
            double ** gapr,
            dcomplex ** gapp )
```

This function computes the product between the geometrical asymmetry parameter and the scattering cross-section, like `aps()`, but for a cluster of spheres. See Eq. (3.16) in Borghese, Denti & Saija (2007).

Parameters

zpv	`double ****`
le	`int`
am0m	`complex double **`
w	`complex double **`
sqk	`double`
gapr	`double **`
gapp	`complex double **`

9.48.1.2 apcra()

```
void apcra (
            double **** zpv,
            const int le,
            dcomplex ** am0m,
            int inpol,
            double sqk,
            double ** gaprm,
            dcomplex ** gappm )
```

This function computes the product between the geometrical asymmetry parameter and the scattering cross-section of a cluster using the random average directions.

Parameters

zpv	`double ****`
le	`int`
am0m	`complex double **`
inpol	`int` Polarization type.
sqk	`double`
gaprm	`double **`
gappm	`complex double **`

9.48.1.3 cdtp()

```
dcomplex cdtp (
            dcomplex z,
            dcomplex * vec_am,
            int i,
            int jf,
            int k,
            int nj,
            np_int istep )
```

This function performs the complex inner product. It is used by `lucin()`.

Parameters

z	`complex double` Starting element.

Parameters

vec_am	`complex double *` Vectorized matrix.
i	`int` Index of first sub-matrix row.
jf	`int` Index of first submatrix column.
k	`int` Index of second sub-matrix row.
nj	`int` Range of first sub-matrix columns.
istep	`np_int` Size of rows in the matrix.

9.48.1.4 cgev()

```
double cgev (
            int ipamo,
            int mu,
            int l,
            int m )
```

Parameters

ipamo	`int`
mu	`int`
l	`int`
m	`int`

Returns

result: `double`

9.48.1.5 cms()

```
void cms (
            dcomplex ** am,
            ParticleDescriptor * c1 )
```

This function constructs the multi-centered M-matrix of the cluster, according to Eq. (5.28) of Borghese, Denti & Saija (2007).

Parameters

am	`complex double **`
c1	`ParticleDescriptor *`

9.48.1.6 crsm1()

```
void crsm1 (
            double vk,
```

```
            double exri,
            ParticleDescriptor * c1 )
```

This function computes the intensity of the scattered field for the cluster, averaged on the orientations. It is invoked for IAVM=1 (ANNOTATION: geometry referred to the meridional plane).

Parameters

vk	`double` Wave number.
exri	`double` External medium refractive index.
c1	`ParticleDescriptor *`

9.48.1.7 ghit()

```
dcomplex ghit (
            int ihi,
            int ipamo,
            int nbl,
            int l1,
            int m1,
            int l2,
            int m2,
            ParticleDescriptor * c1 )
```

This function computes the transfer vector going from N2 to N1, using either Hankel, Bessel or Bessel from origin functions.

Parameters

ihi	`int`
ipamo	`int`
nbl	`int`
l1	`int`
m1	`int`
l2	`int`
m2	`int`
c1	`ParticleDescriptor *` Poiunter to a ParticleDescriptor instance.

9.48.1.8 ghit_d()

```
dcomplex ghit_d (
            int ihi,
            int ipamo,
            int nbl,
            int l1,
            int m1,
            int l2,
            int m2,
            ParticleDescriptor * c1,
            double * rac3j )
```

This function computes the transfer vector going from N2 to N1, using either Hankel, Bessel or Bessel from origin functions.

Parameters

ihi	int
ipamo	int
nbl	int
l1	int
m1	int
l2	int
m2	int
c1	ParticleDescriptor *
rac3j	dcomplex *

9.48.1.9 hjv()

```
void hjv (
            double exri,
            double vk,
            int & jer,
            int & lcalc,
            dcomplex & arg,
            ParticleDescriptor * c1 )
```

This function constructs the Hankel function and the Bessel functions vectors. See page 331 in Borghese, Denti & Saija (2007).

Parameters

exri	double External medium refractive index.
vk	double Wave number.
jer	int & Reference to error code flag.
lcalc	int & Reference to the highest order accounted for in calculation.
arg	complex\<double\> &
c1	ParticleDescriptor *

9.48.1.10 lucin()

```
void lucin (
            dcomplex ** am,
            const np_int nddmst,
            np_int n,
            int & ier )
```

This function performs the inversion of the multi-centered M-matrix through LU decomposition. See Eq. (5.29) in Borghese, Denti & Saija (2007).

Parameters

am	complex double **
nddmst	const int64_t
n	int64_t
ier	int &

9.48.1.11 mextc()

```
void mextc (
            double vk,
            double exri,
            dcomplex ** fsac,
            double ** cextlr,
            double ** cext )
```

This funbction computes the average extinction cross-section starting from the definition of the scattering amplitude. See Sec. 3.2.1 of Borghese, Denti & Saija (2007).

Parameters

vk	`double` Wave number.
exri	`double` External medium refractive index.
fsac	Matrix of complex
cextlr	`double **`
cext	`double **`

9.48.1.12 pcros()

```
void pcros (
            double vk,
            double exri,
            ParticleDescriptor * c1 )
```

This function computes the scattering, absorption and extinction cross-sections of the cluster, together with the Forward Scattering Amplitude.

This function is intended to evaluate the particle cross-section. QUESTIUON: correct?

Parameters

vk	`double` Wave number.
exri	`double` External medium refractive index.
c1	`ParticleDescriptor *`

9.48.1.13 pcrsm0()

```
void pcrsm0 (
            double vk,
            double exri,
            int inpol,
            ParticleDescriptor * c1 )
```

This function computes the orientation-averaged scattering, absorption and extinction cross-sections of the cluster, together with the averaged Forward Scattering Amplitude.

Parameters

vk	`double` Wave number.
exri	`double` External medium refractive index.
inpol	`int` Incident field polarization type.
c1	`ParticleDescriptor *`

9.48.1.14 polar()

```
void polar (
            double x,
            double y,
            double z,
            double & r,
            double & cth,
            double & sth,
            double & cph,
            double & sph )
```

This function performs a conversion from the Cartesian coordinates system to the spherical one. It is used by `sphar()`.

Parameters

x	`double` X-axis Cartesian coordinate.
y	`double` Y-axis Cartesian coordinate.
z	`double` Z-axis Cartesian coordinate.
r	`double &` Reference to radial vector (output value).
cth	`double &` Reference to the cosine of the azimuth coordinate (output value).
sth	`double &` Reference to the sine of the azimuth coordinate (output value).
cph	`double &` Reference to the cosine of the elevation coordinate (output value).
sph	`double &` Reference to the sine of the elevation coordinate (output value).

9.48.1.15 r3j000()

```
void r3j000 (
            int j2,
            int j3,
            double * rac3j )
```

This function calculates the 3j(J,J2,J3;0,0,0) symbol for the Clebsch-Gordan coefficients. See Appendix a.3.1 in Borghese, Denti & Saija (2007).

Parameters

j2	`int`
j3	`int`
rac3j	`double *` Vector of 3j symbols.

9.48.1.16 r3jjr()

```
void r3jjr (
            int j2,
            int j3,
            int m2,
            int m3,
            double * rac3j )
```

This function calculates the 3j(J,J2,J3;-M2-M3,M2,M3) symbol for the Clebsch-Gordan coefficients. See Appendix a.3.1 in Borghese, Denti & Saija (2007).

Parameters

j2	int
j3	int
m2	int
m3	int
rac3j	double * Vector of 3j symbols.

9.48.1.17 r3jjr_d()

```
void r3jjr_d (
            int j2,
            int j3,
            int m2,
            int m3,
            double * rac3j )
```

This function calculates the 3j(J,J2,J3;-M2-M3,M2,M3) symbol for the Clebsch-Gordan coefficients. See Appendix a.3.1 in Borghese, Denti & Saija (2007).

Parameters

j2	int
j3	int
m2	int
m3	int
rac3j	double *

9.48.1.18 r3jmr()

```
void r3jmr (
            int j1,
            int j2,
            int j3,
            int m1,
            double * rac3j )
```

This function calculates the 3j(J,J2,J3;M1,M,-M1-M) symbol for the Clebsch-Gordan coefficients. See Appendix a.3.1 in Borghese, Denti & Saija (2007).

Parameters

j1	`int`
j2	`int`
j3	`int`
m1	`int`
rac3j	`double *` Vector of 3j symbols.

9.48.1.19 raba()

```
void raba (
            int le,
            dcomplex ** am0m,
            dcomplex ** w,
            double ** tqce,
            dcomplex ** tqcpe,
            double ** tqcs,
            dcomplex ** tqcps )
```

This function computes radiation torque on on a cluster of spheres as the result of the difference between the extinction and the scattering contributions for a Cartesian coordinate system, as `rabas()`. See Sec. 4.9 in Borghese, Denti & Saija (2007).

Parameters

le	`int`
am0m	`complex double **`
w	`complex double **`
tqce	`double **`
tqcpe	`complex double **`
tqcs	`double **`
tqcps	`complex double **`

9.48.1.20 rftr()

```
void rftr (
            double * u,
            double * up,
            double * un,
            double * gapv,
            double extins,
            double scatts,
            double & rapr,
            double & cosav,
            double & fp,
            double & fn,
            double & fk,
            double & fx,
            double & fy,
            double & fz )
```

This function computes the Cartesian components of the radiation force exerted on a particle. See Sec. 3.2.1 in Borghese, Denti & Saija (2007).

Parameters

u	`double *`
up	`double *`
un	`double *`
gapv	`double *`
extins	`double`
scatts	`double`
rapr	`double &`
cosav	`double &`
fp	`double &`
fn	`double &`
fk	`double &`
fx	`double &`
fy	`double &`
fz	`double &`

9.48.1.21 scr0()

```
void scr0 (
            double vk,
            double exri,
            ParticleDescriptor * c1 )
```

This function computes the scattering, absorption and extinction cross-sections for the spheres composing the cluster, in terms of Mie coefficients, together with the Forward Scattering Amplitude. See Sec. 4.2.1 in Borghese, Denti & Saija (2007).

Parameters

vk	`double` Wave number
exri	`double` External medium refractive index.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.48.1.22 scr2()

```
void scr2 (
            double vk,
            double vkarg,
            double exri,
            double * duk,
            ParticleDescriptor * c1 )
```

This function computes the scattering amplitude for single spheres constituting an aggregate. See Sec. 4.2.1 in Borghese, Denti & Saija (2007).

Parameters

vk	`double` Wave number.
vkarg	`double` ANNOTATION: Argument of wave number.
exri	`double` External medium refractive index.
duk	`double *` ANNOTATION: variation of unit wave vector.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.48.1.23 str()

```
void str (
            double ** rcf,
            ParticleDescriptor * c1 )
```

This function transforms the Cartesian coordinates of the spheres in an aggregate to radial coordinates, then it calls `sphar()` to calculate the vector of spherical harmonics of the incident field.

Parameters

rcf	`double **` Matrix of sphere configuration fractional radii.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.48.1.24 tqr()

```
void tqr (
            double * u,
            double * up,
            double * un,
            double * tqev,
            double * tqsv,
            double & tep,
            double & ten,
            double & tek,
            double & tsp,
            double & tsn,
            double & tsk )
```

This function computes the radiation torques resulting from the difference between absorption and scattering contributions, like `rabas()`, but for a coordinate system oriented along the wave vector and its orthogonal axis. See Sec. 4.9 in Borghese, Denti & Saija (2007).

Parameters

u	`double *`
up	`double *`
un	`double *`
tqev	`double *`
tqsv	`double *`
tep	`double &`
ten	`double &`

Parameters

tek	double &
tsp	double &
tsn	double &
tsk	double &

9.48.1.25 ztm()

```
void ztm (
            dcomplex ** am,
            ParticleDescriptor * c1 )
```

This function computes the single-centered inverted M-matrix appearing in Eq. (5.28) of Borghese, Denti & Saija (2007).

Parameters

am	complex double **
c1	ParticleDescriptor * Pointer to a ParticleDescriptor instance.

9.49 np_tmcode/src/libnptm/Commons.cpp File Reference

Implementation of the common data structures.

```
#include <cstring>
#include "../include/types.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
```

9.50 np_tmcode/src/libnptm/Configuration.cpp File Reference

Implementation of the configuration classes.

```
#include <cmath>
#include <cstdio>
#include <exception>
#include <fstream>
#include <hdf5.h>
#include <regex>
#include <string>
#include <string.h>
#include "../include/types.h"
#include "../include/errors.h"
#include "../include/List.h"
#include "../include/Parsers.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
#include "../include/file_io.h"
```

Functions

- int * check_overlaps (ScattererConfiguration *sconf, GeometryConfiguration *gconf, const double tolerance)

  Check the presence of overlapping spheres in an aggregate.
- double get_overlap (ScattererConfiguration *sconf, GeometryConfiguration *gconf, const int index_0, const int index_1)

  Get the overlap among two spheres.

9.50.1 Function Documentation

9.50.1.1 check_overlaps()

```
int * check_overlaps (
            ScattererConfiguration * sconf,
            GeometryConfiguration * gconf,
            const double tolerance = 0.0 )
```

The theoretical implementation of the code is based on the assumption that the spherical monomers that compose the aggregate do not overlap in space. Small violations of this assumption do not critically affect the results, but they need to be reported. This function performs a scan of the aggregate and returns a summary of any potential overlap, if detected.

Parameters

sconf	`ScattererConfiguration *` Pointer to a ScattererConfiguration instance.
gconf	`GeometryConfiguration *` Pointer to a GeometryConfiguration instance.
tolerance	`const double` A tolerance value below which overlap is ignored.

Returns

overlaps: `int *` A vector containing the number of overlaps and a list of overlapping sphere indices.

9.50.1.2 get_overlap()

```
double get_overlap (
            ScattererConfiguration * sconf,
            GeometryConfiguration * gconf,
            const int index_0,
            const int index_1 )
```

This function computes the thickness of the compenetration between two spheres, in order to compare it with the current tolerance settings.

Parameters

sconf	`ScattererConfiguration *` Pointer to a ScattererConfiguration instance.
gconf	`GeometryConfiguration *` Pointer to a GeometryConfiguration instance.
index_0	`const int` Index of the first sphere.
index_1	`const int` Index of the second sphere.

Returns

overlap: `double` The thickness of the overlapping layer in meters.

9.51 np_tmcode/src/libnptm/cublas_calls.cpp File Reference

Implementation of the interface with CUBLAS libraries.

```
#include "../include/types.h"
```

9.52 np_tmcode/src/libnptm/file_io.cpp File Reference

Implementation of file I/O operations.

```
#include <exception>
#include <fstream>
#include <regex>
#include <cstring>
#include <string>
#include <hdf5.h>
#include "../include/errors.h"
#include "../include/List.h"
#include "../include/types.h"
#include "../include/file_io.h"
```

9.53 np_tmcode/src/libnptm/inclu_subs.cpp File Reference

C++ implementation of INCLUSION subroutines.

```
#include "../include/types.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
#include "../include/sph_subs.h"
#include "../include/clu_subs.h"
```

Functions

- void cnf (int n, dcomplex z, int nm, dcomplex *csj, dcomplex *csy)

  C++ porting of CNF.
- void exma (dcomplex **am, ParticleDescriptor *c1)

  C++ porting of EXMA.
- void incms (dcomplex **am, double enti, ParticleDescriptor *c1)

  C++ porting of INCMS.
- void indme (int i, int npnt, int npntts, double vk, dcomplex ent, double enti, dcomplex entn, int &jer, int &lcalc, dcomplex &arg, ParticleDescriptor *c1)

  C++ porting of INDME.
- void instr (double **rcf, ParticleDescriptor *c1)

  C++ porting of INSTR.
- void ospv (ParticleDescriptor *c1, double vk, double sze, double exri, dcomplex entn, double enti, int &jer, int &lcalc, dcomplex &arg)

  C++ porting of OSPV.

9.53.1 Function Documentation

9.53.1.1 cnf()

```
void cnf (
            int n,
            dcomplex z,
            int nm,
            dcomplex * csj,
            dcomplex * csy )
```

Parameters

n	`int` Bessel y function order.
z	`dcomplex` Argument of Bessel y function.
nm	`int` Maximum computed order.
csj	`dcomplex *` ANNOTATION: Complex spherical J vector.
csy	`dcomplex *` Complex spherical Bessel functions up to desired order.

9.53.1.2 exma()

```
void exma (
            dcomplex ** am,
            ParticleDescriptor * c1 )
```

Parameters

am	`dcomplex **` Field transition coefficients matrix.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.53.1.3 incms()

```
void incms (
            dcomplex ** am,
            double enti,
            ParticleDescriptor * c1 )
```

Parameters

am	`dcomplex **` Field transition coefficients matrix.
enti	`double` ANNOTATION: imaginary part of coating dielectric function.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.53.1.4 indme()

```
void indme (
            int i,
```

```
            int npnt,
            int npntts,
            double vk,
            dcomplex ent,
            double enti,
            dcomplex entn,
            int & jer,
            int & lcalc,
            dcomplex & arg,
            ParticleDescriptor * c1 )
```

Parameters

i	`int` 1-based sphere configuration index.
npnt	`int` ANNOTATION: Number of non transition layer integration points.
npntts	`int` ANNOTATION: Number of transition layer integrtion points.
vk	`double` Vacuum wave vector magnitude.
ent	`dcomplex` ANNOTATION: coating dielectric function.
enti	`double` ANNOTATION: imaginary part of coating dielectric function.
entn	`dcomplex` ANNOTATION: coating refractive index.
jer	`int &` Error code flag.
lcalc	`int &` Maximum order achieved in calculation.
arg	`dcomplex &` ANNOTATION: argument of calculation.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.53.1.5 instr()

```
void instr (
            double ** rcf,
            ParticleDescriptor * c1 )
```

Parameters

rcf	`double **` Pointer to matrix of fractional radii.
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.53.1.6 ospv()

```
void ospv (
            ParticleDescriptor * c1,
            double vk,
            double sze,
            double exri,
            dcomplex entn,
            double enti,
            int & jer,
            int & lcalc,
            dcomplex & arg )
```

Parameters

c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.
vk	`double` ANNOTATION: vacuum wave number.
sze	`double` ANNOTATION: size.
exri	`double` External medium refractive index.
entn	`dcomplex` Outer sphere refractive index.
enti	`double` Imaginary part of the outer medium refractive index.
jer	`int &` Reference to an integer error flag.
lcalc	`int &` Maximum order achieved in calculation.
arg	`dcomplex` Complex Bessel function argument.

9.54 np_tmcode/src/libnptm/lapack_calls.cpp File Reference

Implementation of the interface with LAPACK libraries.

```
#include "../include/types.h"
```

9.55 np_tmcode/src/libnptm/logging.cpp File Reference

Implementation of the logging system.

```
#include <cstdio>
#include <string>
#include "../include/logging.h"
```

9.56 np_tmcode/src/libnptm/magma_calls.cpp File Reference

Implementation of the interface with MAGMA libraries.

```
#include "../include/types.h"
```

9.57 np_tmcode/src/libnptm/outputs.cpp File Reference

Implementation of the code output format system.

```
#include <cstdio>
#include <exception>
#include <string>
#include <cstring>
#include <hdf5.h>
#include "../include/errors.h"
#include "../include/List.h"
#include "../include/types.h"
#include "../include/Configuration.h"
#include "../include/file_io.h"
#include "../include/outputs.h"
```

9.58 np_tmcode/src/libnptm/Parsers.cpp File Reference

Implementation of the parsing functions.

```
#include <exception>
#include <fstream>
#include <string>
#include "../include/errors.h"
#include "../include/List.h"
#include "../include/Parsers.h"
```

Functions

- std::string * load_file (const std::string &file_name, int *count=0)

  Load a text file as a sequence of strings in memory.

9.58.1 Function Documentation

9.58.1.1 load_file()

```
std::string * load_file (
            const std::string & file_name,
            int * count )
```

The configuration of the field expansion code in FORTRAN uses shared memory access and file I/O operations managed by different functions. Although this approach could be theoretically replicated, it is more convenient to handle input and output to distinct files using specific functions. load_file() helps in the task of handling input such as configuration files or text data structures that need to be loaded entirely. The function performs a line-by line scan of the input file and returns an array of strings that can be later parsed and ingested by the concerned code blocks. An optional pointer to integer allows the function to keep track of the number of file lines that were read, if needed.

Parameters

file_name	`string` The path of the file to be read.
count	`int*` Pointer to an integer recording the number of read lines [OPTIONAL, default=NULL].

Returns

array_lines `string*` An array of strings, one for each input file line.

9.59 np_tmcode/src/libnptm/sph_subs.cpp File Reference

C++ implementation of SPHERE subroutines.

```
#include "../include/types.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
#include "../include/sph_subs.h"
```

Functions

- void aps (double ****zpv, int li, int nsph, ParticleDescriptor *c1, double sqk, double *gaps)

  Compute the asymmetry-corrected scattering cross-section of a sphere.
- void cbf (int n, dcomplex z, int &nm, dcomplex *csj)

  Complex Bessel Function.
- double cg1 (int lmpml, int mu, int l, int m)

  Clebsch-Gordan coefficients.
- void diel (int npntmo, int ns, int i, int ic, double vk, ParticleDescriptor *c1)

  Compute the continuous variation of the refractive index and of its radial derivative.
- void dme (int li, int i, int npnt, int npntts, double vk, double exdc, double exri, ParticleDescriptor *c1, int &jer, int &lcalc, dcomplex &arg, int last_conf)

  Compute Mie scattering coefficients.
- double envj (int n, double x)

  Bessel function calculation control parameters.
- void mmulc (dcomplex *vint, double **cmullr, double **cmul)

  Compute the Mueller Transformation Matrix.
- int msta1 (double x, int mp)

  Starting point for Bessel function magnitude.
- int msta2 (double x, int n, int mp)

  Starting point for Bessel function precision.
- void orunve (double *u1, double *u2, double *u3, int iorth, double torth)

  Compute the amplitude of the orthogonal unit vector.
- void pwma (double *up, double *un, dcomplex *ylm, int inpol, int lw, int isq, ParticleDescriptor *c1)

  Compute incident and scattered field amplitudes.
- void rabas (int inpol, int li, int nsph, ParticleDescriptor *c1, double **tqse, dcomplex **tqspe, double **tqss, dcomplex **tqsps)

  Compute radiation torques on a single sphere in Cartesian coordinates.
- void rbf (int n, double x, int &nm, double sj[ ])

  Real Bessel Function.
- void rkc (int npntmo, double step, dcomplex dcc, double &x, int lpo, dcomplex &y1, dcomplex &y2, dcomplex &dy1, dcomplex &dy2)

  Soft layer radial function and derivative.
- void rkt (int npntmo, double step, double &x, int lpo, dcomplex &y1, dcomplex &y2, dcomplex &dy1, dcomplex &dy2, ParticleDescriptor *c1)

  Transition layer radial function and derivative.
- void rnf (int n, double x, int &nm, double *sy)

  Spherical Bessel functions.
- void sphar (double cosrth, double sinrth, double cosrph, double sinrph, int ll, dcomplex *ylm)

  Spherical harmonics for given direction.
- void sscr0 (dcomplex &tfsas, int nsph, int lm, double vk, double exri, ParticleDescriptor *c1)

  Compute scattering, absorption and extinction cross-sections.
- void sscr2 (int nsph, int lm, double vk, double exri, ParticleDescriptor *c1)

  C++ Compute the scattering amplitude and the scattered field intensity.
- void thdps (int lm, double ****zpv)

  Determine the geometrical asymmetry parameter coefficients.
- void upvmp (double thd, double phd, int icspnv, double &cost, double &sint, double &cosp, double &sinp, double *u, double *up, double *un)

  Compute the unitary vectors onb the polarization plane and its orthogonal direction.
- void upvsp (double *u, double *upmp, double *unmp, double *us, double *upsmp, double *unsmp, double *up, double *un, double *ups, double *uns, double *duk, int &isq, int &ibf, double &scand, double &cfmp, double &sfmp, double &cfsp, double &sfsp)

Compute the unitary vector perpendicular to incident and scattering plane.

- void wmamp (int iis, double cost, double sint, double cosp, double sinp, int inpol, int lm, int idot, int nsph, double ∗arg, double ∗u, double ∗up, double ∗un, ParticleDescriptor ∗c1)

  Compute meridional plane-referred geometrical asymmetry parameter coefficients.
- void wmasp (double cost, double sint, double cosp, double sinp, double costs, double sints, double cosps, double sinps, double ∗u, double ∗up, double ∗un, double ∗us, double ∗ups, double ∗uns, int isq, int ibf, int inpol, int lm, int idot, int nsph, double ∗argi, double ∗args, ParticleDescriptor ∗c1)

  Compute the scattering plane-referred geometrical asymmetry parameter coefficients.

9.59.1 Function Documentation

9.59.1.1 aps()

```
void aps (
            double **** zpv,
            int li,
            int nsph,
            ParticleDescriptor * c1,
            double sqk,
            double * gaps )
```

This function computes the product between the geometrical asymmetry parameter and the scattering cross-section. See Sec. 3.2.1 of Borghese, Denti & Saija (2007).

Parameters

zpv	`double ****` Geometrical asymmetry parameter coefficients matrix.
li	`int` Maximum field expansion order.
nsph	`int` Number of spheres.
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.
sqk	`double`
gaps	`double *` Geometrical asymmetry parameter-corrected cross-section.

9.59.1.2 cbf()

```
void cbf (
            int n,
            dcomplex z,
            int & nm,
            dcomplex * csj )
```

This function computes the complex spherical Bessel funtions j. It uses the auxiliary functions `msta1()` and `msta2()` to determine the starting point for backward recurrence. This is the `CSPHJ` implementation of the `specfun` library.

Parameters

n	`int` Order of the function.
z	`complex double` Argument of the function.
nm	`int &` Highest computed order.
csj	`complex double *` The desired function j.

9.59.1.3 cg1()

```
double cg1 (
            int lmpml,
            int mu,
            int l,
            int m )
```

This function comutes the Clebsch-Gordan coefficients for the irreducible spherical tensors. See Sec. 1.6.3 and Table 1.1 in Borghese, Denti & Saija (2007).

Parameters

lmpml	`int`
mu	`int`
l	`int`
m	`int`

Returns

result: `double` Clebsh-Gordan coefficient.

9.59.1.4 diel()

```
void diel (
            int npntmo,
            int ns,
            int i,
            int ic,
            double vk,
            ParticleDescriptor * c1 )
```

This function implements the continuous variation of the refractive index and of its radial derivative through the materials that constitute the sphere and its surrounding medium. See Sec. 5.2 in Borghese, Denti & Saija (2007).

Parameters

npntmo	`int`
ns	`int`
i	`int`
ic	`int`
vk	`double`
c1	`ParticleDescriptor` * Pointer to a `ParticleDescriptor` data structure.

9.59.1.5 dme()

```
void dme (
            int li,
            int i,
```

```
            int npnt,
            int npntts,
            double vk,
            double exdc,
            double exri,
            ParticleDescriptor * c1,
            int & jer,
            int & lcalc,
            dcomplex & arg,
            int last_conf = 0 )
```

This function determines the L-dependent Mie scattering coefficients a_l and b_l for the cases of homogeneous spheres, radially non-homogeneous spheres and, in case of sphere with dielectric function, sustaining longitudinal waves. See Sec. 5.1 in Borghese, Denti & Saija (2007).

Parameters

li	`int` Maximum field expansion order.
i	`int`
npnt	`int`
npntts	`int`
vk	`double` Wave number in scale units.
exdc	`double` External medium dielectric constant.
exri	`double` External medium refractive index.
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.
jer	`int &` Reference to integer error code variable.
lcalc	`int &` Reference to integer variable recording the maximum expansion order accounted for.
arg	`complex double &`
last_conf	`int` Last sphere configuration (used by CLUSTER)

9.59.1.6 envj()

```
double envj (
            int n,
            double x )
```

This function determines the control parameters required to calculate the Bessel functions up to the desired precision.

Parameters

n	`int`
x	`double`

Returns

result: `double`

9.59.1.7 mmulc()

```
void mmulc (
            dcomplex * vint,
```

```
            double ** cmullr,
            double ** cmul )
```

This function computes the Mueller Transformation Matrix, or Phase Matrix. See Sec. 2.8.1 of Borghese, Denti & Saija (2007).

Parameters

vint	`complex double *`
cmullr	`double **`
cmul	`double **`

9.59.1.8 msta1()

```
int msta1 (
            double x,
            int mp )
```

This function determines the starting point for backward recurrence such that the magnitude of all J and j functions is of the order of 10^{-mp}.

Parameters

x	`double` Absolute value of the argumetn to J or j.
mp	`int` Requested order of magnitude.

Returns

result: `int` The necessary starting point.

9.59.1.9 msta2()

```
int msta2 (
            double x,
            int n,
            int mp )
```

This function determines the starting point for backward recurrence such that all J and j functions have `mp` significant digits.

Parameters

x	`double` Absolute value of the argumetn to J or j.
n	`int` Order of the function.
mp	`int` Requested number of significant digits.

Returns

result: `int` The necessary starting point.

9.59.1.10 orunve()

```
void orunve (
            double * u1,
            double * u2,
            double * u3,
            int iorth,
            double torth )
```

This function computes the amplitude of the orthogonal unit vector for a geometry based either on the scattering plane or on the meridional plane. It is used by `upvsp()` and `upvmp()`. See Sec. 2.7 in Borghese, Denti & Saija (2007).

Parameters

u1	`double *`
u2	`double *`
u3	`double *`
iorth	`int`
torth	`double`

9.59.1.11 pwma()

```
void pwma (
            double * up,
            double * un,
            dcomplex * ylm,
            int inpol,
            int lw,
            int isq,
            ParticleDescriptor * c1 )
```

This function computes the amplitudes of the incident and scattered field on the basis of the multi-polar expansion. See Sec. 4.1.1 in Borghese, Denti and Saija (2007).

Parameters

up	`double *`
un	`double *`
ylm	`complex double *` Field polar spherical harmonics.
inpol	`int` Incident field polarization type (0 - linear; 1 - circular).
lw	`int`
isq	`int`
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.

9.59.1.12 rabas()

```
void rabas (
            int inpol,
            int li,
```

```
            int nsph,
            ParticleDescriptor * c1,
            double ** tqse,
            dcomplex ** tqspe,
            double ** tqss,
            dcomplex ** tqsps )
```

This function computes radiation torque on a sphere unit as the result of the difference between the extinction and the scattering contributions. See Sec. 4.9 in Borghese, Denti & Saija (2007).

Parameters

inpol	`int` Incident polarization type (0 - linear; 1 - circular)
li	`int` Maximum field expansion order.
nsph	`int` Number of spheres.
c1	`ParticleDescriptor *` Pointer to `ParticleDescriptor` data structure.
tqse	`double **`
tqspe	`complex double **`
tqss	`double **`
tqsps	`complex double **`

9.59.1.13 rbf()

```
void rbf (
            int n,
            double x,
            int & nm,
            double sj[] )
```

This function computes the real spherical Bessel funtions j. It uses the auxiliary functions `msta1()` and `msta2()` to determine the starting point for backward recurrence. This is the `SPHJ` implementation of the `specfun` library.

Parameters

n	`int` Order of the function.
x	`double` Argument of the function.
nm	`int &` Highest computed order.
sj	`double[]` The desired function j.

9.59.1.14 rkc()

```
void rkc (
            int npntmo,
            double step,
            dcomplex dcc,
            double & x,
            int lpo,
            dcomplex & y1,
            dcomplex & y2,
```

```
            dcomplex & dy1,
            dcomplex & dy2 )
```

This function determines the radial function and its derivative for a soft layer in a radially non homogeneous sphere. See Sec. 5.1 in Borghese, Denti & Saija (2007).

Parameters

npntmo	`int`
step	`double`
dcc	`complex double`
x	`double &`
lpo	`int`
y1	`complex double &`
y2	`complex double &`
dy1	`complex double &`
dy2	`complex double &`

9.59.1.15 rkt()

```
void rkt (
            int npntmo,
            double step,
            double & x,
            int lpo,
            dcomplex & y1,
            dcomplex & y2,
            dcomplex & dy1,
            dcomplex & dy2,
            ParticleDescriptor * c1 )
```

This function determines the radial function and its derivative for a transition layer in a radially non homogeneous sphere. See Sec. 5.1 in Borghese, Denti & Saija (2007).

Parameters

npntmo	`int`
step	`double`
x	`double &`
lpo	`int`
y1	`complex double &`
y2	`complex double &`
dy1	`complex double &`
dy2	`complex double &`
c1	`ParticleDescriptor *` Pointer to a ParticleDescriptor instance.

9.59.1.16 rnf()

```
void rnf (
            int n,
```

```
            double x,
            int & nm,
            double * sy )
```

This function computes the spherical Bessel functions y. It adopts the SPHJY implementation of the specfun library.

Parameters

n	int Order of the function (from 0 up).
x	double Argumento of the function ($x > 0$).
nm	int & Highest computed order.
sy	double * The desired function y.

9.59.1.17 sphar()

```
void sphar (
            double cosrth,
            double sinrth,
            double cosrph,
            double sinrph,
            int ll,
            dcomplex * ylm )
```

This function computes the field spherical harmonics for a given direction. See Sec. 1.5.2 in Borghese, Denti & Saija (2007).

Parameters

cosrth	double Cosine of direction's elevation.
sinrth	double Sine of direction's elevation.
cosrph	double Cosine of direction's azimuth.
sinrph	double Sine of direction's azimuth.
ll	int L value expansion order.
ylm	complex double * The requested spherical harmonics.

9.59.1.18 sscr0()

```
void sscr0 (
            dcomplex & tfsas,
            int nsph,
            int lm,
            double vk,
            double exri,
            ParticleDescriptor * c1 )
```

This function computes the scattering, absorption and extinction cross-sections in terms of Forward Scattering Amplitudes. See Sec. 4.2.1 in Borghese, Denti & Saija (2007).

Parameters

tfsas	`complex double &`
nsph	`int` Number of spheres.
lm	`int` Maximum field expansion order.
vk	`double` Wave number in scale units.
exri	`double` External medium refractive index.
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.

9.59.1.19 sscr2()

```
void sscr2 (
            int nsph,
            int lm,
            double vk,
            double exri,
            ParticleDescriptor * c1 )
```

The role of this function is to compute the scattering amplitude and the intensity of the scattered field. See Sec. 4.2 in Borghese, Denti & Saija (2007).

Parameters

nsph	`int` Number of spheres.
lm	`int` Maximum field expansion order.
vk	`double` Wave number in scale units.
exri	`double` External medium refractive index.
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.

9.59.1.20 thdps()

```
void thdps (
            int lm,
            double **** zpv )
```

This function computes the coefficients that enter the definition of the geometrical asymmetry parameter based on the L-value of the field expansion order. See Sec. 3.2.1 in Borghese, Denti & Saija (2007).

Parameters

lm	`int` Maximum field expansion order.
zpv	`double ****` Matrix of geometrical asymmetry parameter coefficients.

9.59.1.21 upvmp()

```
void upvmp (
            double thd,
```

```
            double phd,
            int icspnv,
            double & cost,
            double & sint,
            double & cosp,
            double & sinp,
            double * u,
            double * up,
            double * un )
```

This function computes the unitary vectors lying on the polarization plane and on its orthogonal direction, to optimize the identification of the scattering geometry. See Sec. 2.3 in Borghese, Denti & Saija (2007).

Parameters

thd	double
phd	double
icspnv	int
cost	double
sint	double
cosp	double
sinp	double
u	double *
up	double *
un	double *

9.59.1.22 upvsp()

```
void upvsp (
            double * u,
            double * upmp,
            double * unmp,
            double * us,
            double * upsmp,
            double * unsmp,
            double * up,
            double * un,
            double * ups,
            double * uns,
            double * duk,
            int & isq,
            int & ibf,
            double & scand,
            double & cfmp,
            double & sfmp,
            double & cfsp,
            double & sfsp )
```

This function computes the unitary vector perpendicular to the incident and scattering plane in a geometry based on the scattering plane. It uses `orunve()`. See Sec. 2.7 in Borghese, Denti & Saija (2007).

Parameters

u	double *

Parameters

upmp	`double *`
unmp	`double *`
us	`double *`
upsmp	`double *`
unsmp	`double *`
up	`double *`
un	`double *`
ups	`double *`
uns	`double *`
duk	`double *`
isq	`int &`
ibf	`int &`
scand	`double &` Scattering angle in degrees.
cfmp	`double &`
sfmp	`double &`
cfsp	`double &`
sfsp	`double &`

9.59.1.23 wmamp()

```
void wmamp (
            int iis,
            double cost,
            double sint,
            double cosp,
            double sinp,
            int inpol,
            int lm,
            int idot,
            int nsph,
            double * arg,
            double * u,
            double * up,
            double * un,
            ParticleDescriptor * c1 )
```

This function computes the coeffcients that define the geometrical symmetry parameter as defined with respect to the meridional plane. It makes use of `sphar()` and `pwma()`. See Sec. 3.2.1 in Borghese, Denti & Saija (2007).

Parameters

iis	`int`
cost	`double` Cosine of the elevation angle.
sint	`double` Sine of the elevation angle.
cosp	`double` Cosine of the azimuth angle.
sinp	`double` Sine of the azimuth angle.
inpol	`int` Incident field polarization type (0 - linear; 1 - circular).
lm	`int` Maximum field expansion orde.
idot	`int`

Parameters

nsph	`int` Number of spheres.
arg	`double *`
u	`double *`
up	`double *`
un	`double *`
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.

9.59.1.24 wmasp()

```
void wmasp (
            double cost,
            double sint,
            double cosp,
            double sinp,
            double costs,
            double sints,
            double cosps,
            double sinps,
            double * u,
            double * up,
            double * un,
            double * us,
            double * ups,
            double * uns,
            int isq,
            int ibf,
            int inpol,
            int lm,
            int idot,
            int nsph,
            double * argi,
            double * args,
            ParticleDescriptor * c1 )
```

This function computes the coefficients that define the geometrical asymmetry parameter based on the L-value with respect to the scattering plane. It uses `sphar()` and `pwma()`. See Sec. 3.2.1 in Borghese, Denti and Saija (2007).

Parameters

cost	`double` Cosine of elevation angle.
sint	`double` Sine of elevation angle.
cosp	`double` Cosine of azimuth angle.
sinp	`double` Sine of azimuth angle.
costs	`double` Cosine of scattering elevation angle.
sints	`double` Sine of scattering elevation angle.
cosps	`double` Cosine of scattering azimuth angle.
sinps	`double` Sine of scattering azimuth angle.
u	`double *`
up	`double *`
un	`double *`

Parameters

us	`double *`
ups	`double *`
uns	`double *`
isq	`int`
ibf	`int`
inpol	`int` Incident field polarization (0 - linear; 1 -circular).
lm	`int` Maximum field expansion order.
idot	`int`
nsph	`int` Number opf spheres.
argi	`double *`
args	`double *`
c1	`ParticleDescriptor *` Pointer to a `ParticleDescriptor` data structure.

9.60 np_tmcode/src/libnptm/tfrfme.cpp File Reference

Implementation of the trapping calculation objects.

```
#include <exception>
#include <fstream>
#include <hdf5.h>
#include <string>
#include "../include/types.h"
#include "../include/errors.h"
#include "../include/List.h"
#include "../include/tfrfme.h"
#include "../include/file_io.h"
```

9.61 np_tmcode/src/libnptm/tra_subs.cpp File Reference

C++ implementation of TRAPPING subroutines.

```
#include <cmath>
#include <fstream>
#include "../include/types.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
#include "../include/sph_subs.h"
#include "../include/tfrfme.h"
#include "../include/tra_subs.h"
```

Functions

- void camp (dcomplex *ac, dcomplex **am0m, dcomplex *ws, CIL *cil)
- void czamp (dcomplex *ac, dcomplex **amd, int **indam, dcomplex *ws, CIL *cil)
- void ffrf (double ****zpv, dcomplex *ac, dcomplex *ws, double *fffe, double *fffs, CIL *cil, CCR *ccr)
- void ffrt (dcomplex *ac, dcomplex *ws, double *ffte, double *ffts, CIL *cil)
- dcomplex * frfmer (int nkv, double vkm, double vknmx, double apfafa, double tra, double spd, double rir, double ftcn, int le, int lmode, double pmf, Swap1 *tt1, Swap2 *tt2)
- void pwmalp (dcomplex **w, double *up, double *un, dcomplex *ylm, int lw)
- void samp (dcomplex *ac, dcomplex *tmsm, dcomplex *tmse, dcomplex *ws, CIL *cil)
- void sampoa (dcomplex *ac, dcomplex **tms, dcomplex *ws, CIL *cil)
- void wamff (dcomplex *wk, double x, double y, double z, int lm, double apfafa, double tra, double spd, double rir, double ftcn, int lmode, double pmf)

9.61.1 Function Documentation

9.61.1.1 camp()

```
void camp (
            dcomplex * ac,
            dcomplex ** am0m,
            dcomplex * ws,
            CIL * cil )
```

C++ porting of CAMP

Parameters

ac	`complex double *` ANNOTATION: vector of complex amplitudes.
am0m	`complex double **` ANNOTATION: T-matrix of the particle.
ws	`complex double *` ANNOTATION: scattering amplitudes.
cil	`CIL *` Pointer to a CIL structure.

This function builds the AC vector using AM0M and WS.

9.61.1.2 czamp()

```
void czamp (
            dcomplex * ac,
            dcomplex ** amd,
            int ** indam,
            dcomplex * ws,
            CIL * cil )
```

C++ porting of CZAMP

Parameters

ac	Vector of complex. ANNOTATION: vector of complex amplitudes.
amd	Matrix of complex. ANNOTATION: T-matrix.
indam	`int **`. ANNOTATION: map of matrix indexes.
ws	Vector of complex. ANNOTATION: scattering amplitudes.
cil	`CIL *` Pointer to a CIL structure.

This function builds the AC vector using AMD, INDAM and WS.

9.61.1.3 ffrf()

```
void ffrf (
            double **** zpv,
            dcomplex * ac,
            dcomplex * ws,
            double * fffe,
            double * fffs,
            CIL * cil,
            CCR * ccr )
```

C++ porting of FFRF

Parameters

zpv	`double ****`. ANNOTATION: asymmetry tensor.
ac	`complex double *` ANNOTATION: vector of complex amplitudes.
ws	`complex double *` ANNOTATION: scattering amplitudes.
fffe	`double *`. ANNOTATION: extinction contributions to force.
fffs	`double *`. ANNOTATION: scattering contributions to force.
cil	`CIL *` Pointer to a CIL structure.
ccr	`CCR *` Pointer to a CCR structure.

9.61.1.4 ffrt()

```
void ffrt (
            dcomplex * ac,
            dcomplex * ws,
            double * ffte,
            double * ffts,
            CIL * cil )
```

C++ porting of FFRT

Parameters

ac	`complex double *` ANNOTATION: vector of complex amplitudes.
ws	`complex double *` ANNOTATION: scattering amplitudes.
ffte	`double *`. ANNOTATION: extinction contributions to torque.
ffts	`double *`. ANNOTATION: scattering contributions to torque.
cil	`CIL *` Pointer to a CIL structure.

9.61.1.5 frfmer()

```
dcomplex * frfmer (
            int nkv,
```

```
            double vkm,
            double vknmx,
            double apfafa,
            double tra,
            double spd,
            double rir,
            double ftcn,
            int le,
            int lmode,
            double pmf,
            Swap1 * tt1,
            Swap2 * tt2 )
```

C++ porting of FRFMER

Parameters

nkv	`int` ANNOTATION: number of wave vectors.
vkm	`double` ANNOTATION: wave number.
vknmx	`double` ANNOTATION: maximum wave number.
apfafa	`double` ANNOTATION: normalized aperture.
tra	`double` ANNOTATION: lens transmission.
spd	`double` ANNOTATION: cover thickness.
rir	`double` ANNOTATION: external over oil refractive index ratio.
ftcn	`double` ANNOTATION: refraction factor.
le	`int` ANNOTATION: external multipole truncation order.
lmode	`int` ANNOTATION: laser mode.
pmf	`double` ANNOTATION: aperture correction.
tt1	`Swap1 *` Pointer to first swap object.
tt2	`Swap2 *` Pointer to second swap object.

Returns

wk: `complex double *` ANNOTATION: incident amplitudes.

9.61.1.6 pwmalp()

```
void pwmalp (
            dcomplex ** w,
            double * up,
            double * un,
            dcomplex * ylm,
            int lw )
```

C++ porting of PWMALP

Parameters

w	`complex double *` ANNOTATION: amplitudes.
up	`double *` ANNOTATION: unit polarization vector.
un	`double *` ANNOTATION: unit normal vector.
ylm	`complex double *` Field vector spherical harmonics.
lw	`int` ANNOTATION: order of amplitudes.

9.61.1.7 samp()

```
void samp (
            dcomplex * ac,
            dcomplex * tmsm,
            dcomplex * tmse,
            dcomplex * ws,
            CIL * cil )
```

C++ porting of SAMP

Parameters

ac	complex double * ANNOTATION: vector of complex amplitudes.
tmsm	complex double * ANNOTATION: T-matrix M components.
tmse	complex double * ANNOTATION: T-matrix E components.
ws	complex double * ANNOTATION: scattering amplitudes.
cil	CIL * Pointer to a CIL structure.

This function builds the AC vector using TMSM, TMSE and WS.

9.61.1.8 sampoa()

```
void sampoa (
            dcomplex * ac,
            dcomplex ** tms,
            dcomplex * ws,
            CIL * cil )
```

C++ porting of SAMPOA

Parameters

ac	complex double * ANNOTATION: vector of complex amplitudes.
tms	complex double ** ANNOTATION: T-matrix.
ws	complex double * ANNOTATION: scattering amplitudes.
cil	CIL * Pointer to a CIL structure.

This function builds the AC vector using TMS and WS.

9.61.1.9 wamff()

```
void wamff (
            dcomplex * wk,
            double x,
            double y,
            double z,
            int lm,
            double apfafa,
            double tra,
```

```
            double spd,
            double rir,
            double ftcn,
            int lmode,
            double pmf )
```

C++ porting of WAMFF

Parameters

wk	`complex double *` ANNOTATION: incident amplitudes.
x	`double` ANNOTATION: X coordinate.
y	`double` ANNOTATION: Y coordinate.
z	`double` ANNOTATION: Z coordinate.
lm	`int` ANNOTATION: maximum multipole truncation order.
apfafa	`double` ANNOTATION: Normalized aperture.
tra	`double` ANNOTATION: Lens transmission.
spd	`double` ANNOTATION: Cover slip thickness.
rir	`double` ANNOTATION: External over oil refractive index ratio.
ftcn	`double` ANNOTATION: Refraction factor.
lmode	`int` ANNOTATION: Laser mode.
pmf	`double` ANNOTATION: aperture correction.

9.62 np_tmcode/src/libnptm/TransitionMatrix.cpp File Reference

Implementation of the Transition Matrix structure.

```
#include <exception>
#include <fstream>
#include <hdf5.h>
#include <string>
#include "../include/types.h"
#include "../include/errors.h"
#include "../include/List.h"
#include "../include/TransitionMatrix.h"
#include "../include/file_io.h"
```

9.63 np_tmcode/src/libnptm/utils.cpp File Reference

Implementation of auxiliary code utilities.

```
#include <hdf5.h>
#include "../include/types.h"
#include "../include/errors.h"
#include "../include/List.h"
#include "../include/file_io.h"
#include "../include/utils.h"
```

Functions

- double get_ram_overhead ()

  Obtain an estimate of the RAM overhead factor for optimization.
- int write_dcomplex_matrix (VirtualAsciiFile *af, dcomplex **mat, int rows, int columns, const std::string &format, int first_index)

  Write a double complex matrix to a text file.

9.63.1 Function Documentation

9.63.1.1 get_ram_overhead()

```
double get_ram_overhead ( )
```

Code speed-up optimization usually comes at the cost of increased memory requirements. While this should not generally be a serious issue, it can become such in case of models that require large amounts of memory to be handled. This function tests the code build configuration to provide an zero-order estimate of the weight of these overheads to protect the host system from saturating the RAM.

Returns

factor: `double` The multiplicative factor to be applied to data size.

9.63.1.2 write_dcomplex_matrix()

```
int write_dcomplex_matrix (
            VirtualAsciiFile * af,
            dcomplex ** mat,
            int rows,
            int columns,
            const std::string & format = " %5d %5d (%17.8lE,%17.8lE)\n",
            int first_index = 1 )
```

Parameters

af	`VirtualAsciiFile *` Pointer to an existing VirtualAsciiFile.
mat	`dcomplex **` Pointer to the matrix.
rows	`int` Number of rows in the matrix.
columns	`int` Number of columns in the matrix.
format	`const string&` Format of the line (default is " %5d %5d (%17.8lE,%17.8lE) ")
first_index	`int` Index of the first element (default is 1, i.e. base 1 FORTRAN array notation)

9.64 np_tmcode/src/sphere/np_sphere.cpp File Reference

Single sphere scattering problem handler.

```
#include <cstdio>
#include <string>
#include "../include/types.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
```

Functions

- void sphere (const string &config_file, const string &data_file, const string &output_path, const mixMPI *mpidata)

  C++ implementation of SPH.
- int main (int argc, char **argv)

  Main program entry point.

9.64.1 Detailed Description

This program emulates the execution work-flow originally handled by the FORTRAN EDFB and SPH codes, which undertook the task of solving the scattering calculation for the case of a single, possibly multi-layered sphere. The program is designed to work in two modes: emulation and enhanced. The emulation mode is activated by invoking the program without arguments. In this case, the code looks for hard-coded default input and writes output in the execution folder, replicating the behaviour of the original FORTRAN code. Advanced mode, instead, is activated by passing command line arguments that locate the desired input files and a valid folder to write the output into. The emulation mode is useful for testing, while the advanced mode implements the possibility to change input and output options, without having to modify the code.

9.64.2 Function Documentation

9.64.2.1 main()

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

This is the starting point of the execution flow. Here we may choose how to configure the code, e.g. by loading a legacy configuration file or some otherwise formatted configuration data set. The code can be linked to a luncher script or to a GUI oriented application that performs the configuration and runs the main program.

Parameters

argc	`int` The number of arguments given in command-line.
argv	`char **` The vector of command-line arguments.

Returns

result: `int` An exit code passed to the OS (0 for succesful execution).

9.64.2.2 sphere()

```
void sphere (
```

```
            const string & config_file,
            const string & data_file,
            const string & output_path,
            const mixMPI * mpidata )  [extern]
```

Parameters

config_file	`string` Name of the configuration file.
data_file	`string` Name of the input data file.
output_path	`string` Directory to write the output files in.
mpidata	`const mixMPI *` Pointer to a mixMPI data structure.

9.65 np_tmcode/src/sphere/sphere.cpp File Reference

Implementation of the single sphere calculation.

```
#include <cstdio>
#include <exception>
#include <fstream>
#include <hdf5.h>
#include <string>
#include "../include/types.h"
#include "../include/errors.h"
#include "../include/logging.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
#include "../include/sph_subs.h"
#include "../include/TransitionMatrix.h"
#include "../include/List.h"
#include "../include/file_io.h"
#include "../include/outputs.h"
#include "../include/IterationData.h"
```

Functions

- int sphere_jxi488_cycle (int jxi488, ScattererConfiguration *sconf, GeometryConfiguration *gconf, ScatteringAngles *sa, SphereIterationData *sid, SphereOutputInfo *oi, const string &output_path, VirtualBinaryFile *vtppoanp)

  Main calculation loop.
- void sphere (const string &config_file, const string &data_file, const string &output_path, const mixMPI *mpidata)

  C++ implementation of SPH.

9.65.1 Function Documentation

9.65.1.1 sphere()

```
void sphere (
            const string & config_file,
            const string & data_file,
            const string & output_path,
            const mixMPI * mpidata )
```

Parameters

config_file	`string` Name of the configuration file.
data_file	`string` Name of the input data file.
output_path	`string` Directory to write the output files in.
mpidata	`const mixMPI *` Pointer to a mixMPI data structure.

9.65.1.2 sphere_jxi488_cycle()

```
int sphere_jxi488_cycle (
            int jxi488,
            ScattererConfiguration * sconf,
            GeometryConfiguration * gconf,
            ScatteringAngles * sa,
            SphereIterationData * sid,
            SphereOutputInfo * oi,
            const string & output_path,
            VirtualBinaryFile * vtppoanp )
```

Parameters

jxi488	`int` Wavelength loop index.
sconf	`ScattererConfiguration *` Pointer to a `ScattererConfiguration` object.
gconf	`GeometryConfiguration *` Pointer to a `GeometryConfiguration` object.
sa	`ScatteringAngles *` Pointer to a `ScatteringAngles` object.
sid	`SphereIterationData *` Pointer to a `SphereIterationData` object.
oi	`SphereOutputInfo *` Pointer to a `SphereOutputInfo` object.
output_path	`const string &` Path to the output directory.
vtppoanp	`VirtualBinaryFile *` Pointer to a `VirtualBinaryFile` object.

9.66 np_tmcode/src/testing/test_file_io.cpp File Reference

```
#include <cstdio>
#include <exception>
#include <hdf5.h>
#include <string>
#include "../include/types.h"
#include "../include/errors.h"
#include "../include/List.h"
#include "../include/file_io.h"
```

Functions

- int **main** ()

  Main program execution body.

9.67 np_tmcode/src/testing/test_TEDF.cpp File Reference

```
#include <cstdio>
#include <exception>
#include <hdf5.h>
#include <string>
#include "../include/types.h"
#include "../include/errors.h"
#include "../include/List.h"
#include "../include/file_io.h"
#include "../include/Configuration.h"
```

Functions

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

  Main program execution body.

9.67.1 Function Documentation

9.67.1.1 main()

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

This program executes a test to compare whether three configuration instances, loaded respectively from an EDFB configuration file, a legacy binary, or a HDF5 binary are actually equivalent. The test writes a result message to `stdout` then it returns 0 (OS flag for a successful process) or some kind of error code, depending on whether the test files were found all equal or not. The test accepts three command line arguments: the name of the EDFB configuration file, the name of the legacy binary file and the name of the HDF5 binary configuration file.

Parameters

argc	`int` Number of command line arguments
argv	`char **` Array of command argument character strings.

Returns

result: `int` Can be: 0 (all files equal); 1 (EDFB and legacy binary are different); 10 (EDFB and HDF5 are different); 100 (legacy and HDF5 are different). In case more differences are found, the error codes sum up together (e.g. 111 means all files are different).

9.68 np_tmcode/src/testing/test_TTMS.cpp File Reference

```
#include <cstdio>
#include <exception>
#include <hdf5.h>
#include <string>
```

```
#include "../include/types.h"
#include "../include/errors.h"
#include "../include/List.h"
#include "../include/file_io.h"
#include "../include/TransitionMatrix.h"
```

Functions

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

  Main program execution body.

9.68.1 Function Documentation

9.68.1.1 main()

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

This program executes a test to compare whether two transition matrix instances, loaded respectively from a legacy and a HDF5 binary file are actually equivalent. The test writes a result message to `stdout` then it returns 0 (OS flag for a successful process) or 1 (OS flag for failing process) depending on whether the two instances were considered equivalent or not.

Parameters

argc	`int` Number of command line arguments
argv	`char **` Array of command argument character strings.

Returns

result: `int` Can be 0 (files are equal) or 1 (files are different).

9.69 np_tmcode/src/trapping/cfrfme.cpp File Reference

```
#include <chrono>
#include <cstdio>
#include <exception>
#include <fstream>
#include <regex>
#include <string>
#include "../include/types.h"
#include "../include/Parsers.h"
#include "../include/logging.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
#include "../include/sph_subs.h"
#include "../include/tfrfme.h"
#include "../include/tra_subs.h"
#include "../include/errors.h"
```

Functions

- void frfme (string data_file, string output_path)
  C++ implementation of FRFME.

9.69.1 Detailed Description

C++ implementation of FRFME functions.

9.69.2 Function Documentation

9.69.2.1 frfme()

```
void frfme (
            string data_file,
            string output_path )
```

Parameters

data_file	`string` Name of the input data file.
output_path	`string` Directory to write the output files in.

9.70 np_tmcode/src/trapping/clffft.cpp File Reference

C++ implementation of LFFFT functions.

```
#include <chrono>
#include <cmath>
#include <cstdio>
#include <exception>
#include <fstream>
#include <hdf5.h>
#include <regex>
#include <string>
#include "../include/types.h"
#include "../include/Parsers.h"
#include "../include/logging.h"
#include "../include/Configuration.h"
#include "../include/Commons.h"
#include "../include/sph_subs.h"
#include "../include/tfrfme.h"
#include "../include/tra_subs.h"
#include "../include/errors.h"
#include "../include/List.h"
#include "../include/TransitionMatrix.h"
#include "../include/file_io.h"
#include "../include/outputs.h"
```

Functions

- void lffft (string data_file, string output_path)
  C++ implementation of LFFFT.

9.70.1 Function Documentation

9.70.1.1 lffft()

```
void lffft (
            string data_file,
            string output_path )
```

Parameters

data_file	`string` Name of the input data file.
output_path	`string` Directory to write the output files in.

9.71 np_tmcode/src/trapping/np_trapping.cpp File Reference

Trapping problem handler.

```
#include <chrono>
#include <cstdio>
#include <string>
#include "../include/logging.h"
```

Functions

- void frfme (string data_file, string output_path)
  C++ implementation of FRFME.
- void lffft (string data_file, string output_path)
  C++ implementation of LFFFT.
- int main (int argc, char **argv)
  Main program execution body.

9.71.1 Function Documentation

9.71.1.1 frfme()

```
void frfme (
            string data_file,
            string output_path )  [extern]
```

Parameters

data_file	`string` Name of the input data file.
output_path	`string` Directory to write the output files in.

9.71.1.2 lffft()

```
void lffft (
            string data_file,
            string output_path )  [extern]
```

Parameters

data_file	`string` Name of the input data file.
output_path	`string` Directory to write the output files in.

9.71.1.3 main()

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

Parameters

argc	`int`
argv	`char **`

Returns

result: `int`

Index