

oepdev
0.0.0

Generated by Doxygen 1.8.6

Fri Feb 2 2018 19:36:02

Contents

1	Main Page	1
2	License	3
3	OEP Design.	5
3.1	OEP Classes	5
3.1.1	Structure of possible OEP-based expressions and their unification	5
4	List of One-Electron Potentials	7
4.1	Electrostatic Energy OEP's	7
4.2	Pauli Repulsion OEP's	7
4.2.1	First-order contribution in overlap matrix expansion.	7
4.2.2	Second-order contribution in overlap matrix expansion.	8
4.3	Excitonic Energy Transfer OEP's	8
4.3.1	ET contributions.	8
4.3.2	HT contributions.	8
4.3.3	CT contributions.	8
4.4	Full HF Interaction OEP's	8
5	Density-fitting specialized for OEP's	9
6	Implemented Models	11
6.1	Target Properties	11
6.2	Target, Benchmark and Competing Models	11
7	Contributing to oep-dev	13
7.1	Main routine and libraries	13
7.2	Header files in libraries	13
7.3	Environmental variables	14
7.4	Documenting the code	14
7.5	Naming conventions	14
7.6	Use Object-Oriented Programming	15
8	Namespace Index	17

8.1	Namespace List	17
9	Hierarchical Index	19
9.1	Class Hierarchy	19
10	Class Index	21
10.1	Class List	21
11	Namespace Documentation	23
11.1	oepdev Namespace Reference	23
11.1.1	Detailed Description	24
11.1.2	Function Documentation	25
11.1.2.1	create_superfunctional	25
11.1.2.2	extract_monomer	25
11.1.2.3	solve_scf	25
11.2	psi Namespace Reference	25
11.2.1	Detailed Description	26
11.2.2	Function Documentation	26
11.2.2.1	oepdev	26
11.2.2.2	read_options	27
12	Class Documentation	29
12.1	oepdev::AllAOIntegralsIterator Class Reference	29
12.1.1	Detailed Description	29
12.1.2	Constructor & Destructor Documentation	30
12.1.2.1	AllAOIntegralsIterator	30
12.1.2.2	AllAOIntegralsIterator	30
12.1.3	Member Function Documentation	30
12.1.3.1	index	30
12.2	oepdev::AllAOShellCombinationsIterator Class Reference	30
12.2.1	Detailed Description	31
12.2.2	Constructor & Destructor Documentation	31
12.2.2.1	AllAOShellCombinationsIterator	31
12.2.2.2	AllAOShellCombinationsIterator	31
12.2.3	Member Function Documentation	32
12.2.3.1	compute_shell	32
12.3	oepdev::CPHF Class Reference	32
12.3.1	Detailed Description	33
12.3.2	Constructor & Destructor Documentation	33
12.3.2.1	CPHF	33
12.4	oepdev::CubePoints3DIterator Class Reference	34

12.4.1 Detailed Description	34
12.5 oepdev::CubePointsCollection3D Class Reference	35
12.5.1 Detailed Description	35
12.6 oepdev::DIISManager Class Reference	35
12.6.1 Detailed Description	36
12.6.2 Constructor & Destructor Documentation	36
12.6.2.1 DIISManager	36
12.6.3 Member Function Documentation	36
12.6.3.1 compute	36
12.6.3.2 put	36
12.6.3.3 update	36
12.7 oepdev::EETCouplingOEPotential Class Reference	36
12.7.1 Detailed Description	37
12.7.2 Member Function Documentation	37
12.7.2.1 compute_3D	37
12.8 oepdev::ElectrostaticEnergyOEPotential Class Reference	37
12.8.1 Detailed Description	38
12.8.2 Member Function Documentation	38
12.8.2.1 compute_3D	38
12.9 oepdev::ElectrostaticEnergySolver Class Reference	38
12.9.1 Detailed Description	39
12.9.2 Member Function Documentation	40
12.9.2.1 compute_benchmark	40
12.9.2.2 compute_oep_based	40
12.10 oepdev::ElectrostaticPotential3D Class Reference	41
12.10.1 Detailed Description	41
12.11 oepdev::ESPSolver Class Reference	42
12.11.1 Detailed Description	42
12.11.2 Constructor & Destructor Documentation	43
12.11.2.1 ESPSolver	43
12.11.2.2 ESPSolver	43
12.12 oepdev::OEPDevSolver Class Reference	43
12.12.1 Detailed Description	44
12.12.2 Constructor & Destructor Documentation	44
12.12.2.1 OEPDevSolver	44
12.12.3 Member Function Documentation	44
12.12.3.1 build	44
12.12.3.2 compute_benchmark	45
12.12.3.3 compute_oep_based	45
12.13 oepdev::OEPotential Class Reference	45

12.13.1 Detailed Description	47
12.13.2 Constructor & Destructor Documentation	47
12.13.2.1 OEPotential	47
12.13.2.2 OEPotential	47
12.13.3 Member Function Documentation	48
12.13.3.1 build	48
12.13.3.2 build	48
12.13.3.3 compute_3D	48
12.13.3.4 write_cube	48
12.14 oepdev::OEPotential3D< T > Class Template Reference	48
12.14.1 Detailed Description	49
12.14.2 Constructor & Destructor Documentation	50
12.14.2.1 OEPotential3D	50
12.14.2.2 OEPotential3D	50
12.15 oepdev::Points3DIterator::Point Struct Reference	50
12.16 oepdev::Points3DIterator Class Reference	51
12.16.1 Detailed Description	52
12.16.2 Constructor & Destructor Documentation	52
12.16.2.1 Points3DIterator	52
12.16.3 Member Function Documentation	52
12.16.3.1 build	52
12.16.3.2 build	53
12.16.3.3 build	54
12.17 oepdev::PointsCollection3D Class Reference	54
12.17.1 Detailed Description	55
12.17.2 Constructor & Destructor Documentation	55
12.17.2.1 PointsCollection3D	55
12.17.3 Member Function Documentation	56
12.17.3.1 build	56
12.17.3.2 build	56
12.17.3.3 build	56
12.18 oepdev::PotentialInt Class Reference	56
12.18.1 Detailed Description	57
12.18.2 Constructor & Destructor Documentation	57
12.18.2.1 PotentialInt	57
12.18.2.2 PotentialInt	57
12.18.2.3 PotentialInt	58
12.18.3 Member Function Documentation	58
12.18.3.1 set_charge_field	58
12.19 oepdev::RandomPoints3DIterator Class Reference	58

12.19.1 Detailed Description	59
12.20oepdev::RandomPointsCollection3D Class Reference	60
12.20.1 Detailed Description	60
12.21oepdev::RepulsionEnergyOEPotential Class Reference	60
12.21.1 Detailed Description	61
12.21.2 Member Function Documentation	61
12.21.2.1 compute_3D	61
12.22oepdev::RepulsionEnergySolver Class Reference	61
12.22.1 Detailed Description	62
12.22.2 Member Function Documentation	64
12.22.2.1 compute_benchmark	64
12.22.2.2 compute_oep_based	64
12.23oepdev::ScalarField3D Class Reference	64
12.23.1 Detailed Description	66
12.23.2 Member Function Documentation	66
12.23.2.1 build	66
12.23.2.2 build	67
12.24oepdev::WavefunctionUnion Class Reference	67
12.24.1 Detailed Description	70
12.24.2 Constructor & Destructor Documentation	71
12.24.2.1 WavefunctionUnion	71
12.24.3 Member Function Documentation	71
12.24.3.1 Ca_subset	71
12.24.3.2 Cb_subset	71
Index	73

Chapter 1

Main Page

oep-dev

Generalized One-Electron Potentials: Development Platform.

Contact: Bartosz Blasiak (blasiak.bartosz@gmail.com)

Overview

Test various models of the intermolecular interaction that is based on the application of the **One-Electron Potentials (OEP's)** technique.

Currently, the interaction between two molecules described by the Hartree-Fock-Roothaan-Hall theory or the configuration interaction with singles theory is considered. In particular, the plugin tests the models of:

1. the Pauli exchange-repulsion interaction energy (Project II)
2. the Induction interaction energy (Project III)
3. the excitation energy transfer couplings (Project I)

against reference solutions (exact or other approximations).

Places to go:

- https://github.com/globulion/oepdev/blob/master/doc/git/doc_oep_design.md "OEP Design"
- https://github.com/globulion/oepdev/blob/master/doc/git/doc_implemented_models.md "Implemented Models"
- https://github.com/globulion/oepdev/blob/master/doc/git/doc_programming_etiquette.md "Programming Etiquette"
- [Current Issues](#)

References

Chapter 2

License

Copyright (c) 2018, Bartosz Błasiak

All rights reserved.

Usage, copy or redistribution is allowed only after obtaining written consent of the Repository Administrator.

Chapter 3

OEP Design.

OEP (One-Electron Potential) is associated with certain quantum one-electron operator \hat{v}^A that defines the ability of molecule A to interact in a particular way with other molecules.

Technically, OEP can be understood as a **container object** (associated with the molecule in question) that stores the information about the above mentioned quantum operator. Here, it is assumed that similar OEP object is also defined for all other molecules in a molecular aggregate.

In case of interaction between molecules A and B , OEP object of molecule A interacts directly with wavefunction object of the molecule B . Defining a Solver class that handles such interaction Wavefunction class and OEP class the universal design of OEP-based approaches can be established and developed.

Important: OEP and Wavefunction classes should not be restricted to Hartree-Fock; in general any correlated wavefunction and derived OEP's should be allowed to work with each other.

3.1 OEP Classes

There are many types of OEP's, but the underlying principle is the same and independent of the type of intermolecular interaction. Therefore, the OEP's should be implemented by using a multi-level class design. In turn, this design depends on the way OEP's enter the mathematical expressions, i.e., on the types of matrix elements of the one-electron effective operator \hat{v}^A .

3.1.1 Structure of possible OEP-based expressions and their unification

Structure of OEP-based mathematical expressions is listed below:

Type	Matrix Element	Comment
Type 1	$(I \hat{v}^A J)$	$I \in A, J \in B$
Type 2	$(J \hat{v}^A L)$	$J, L \in B$

In the above table, I , J and K indices correspond to basis functions or molecular orbitals. Basis functions can be primary or auxiliary OEP-specialized density-fitting. Depending on the type of function and matrix element, there are many subtypes of resulting matrix elements that differ in their dimensionality. Examples are given below:

Matrix Element	DF-based form	ESP-based form
$(\mu \hat{v}^{A[\mu]} \sigma)$	$\sum_{I \in A} v_{\mu I}^A S_{I\sigma}$	$\sum_{\alpha \in A} q_{\alpha}^{A[\mu]} V_{\mu\sigma}^{(\alpha)}$
$(i \hat{v}^{A[i]} j)$	$\sum_{I \in A} v_{ii}^A S_{Ij}$	$\sum_{\alpha \in A} q_{\alpha}^{A[i]} V_{ij}^{(\alpha)}$

$\left(j \hat{v}^{A[i]} l\right)$	$\sum_{l,k \in A} S_{jl} v_{lk}^{A[i]} S_{kl}$	$\sum_{\alpha \in A} q_{\alpha}^{A[i]} V_{jl}^{(\alpha)}$
-----------------------------------	--	---

In the formulae above, the OEP-part (stored by OEP instances) and the Solver-part (to be computed by the Solver) are separated. It is apparent that all OEP-parts have the form of 2nd- or 3rd-rank tensors with different class of axes (molecular orbitals, primary/auxiliary basis, atomic space). Therefore, they can be uniquely defined by a unified *tensor object* (storing double precision numbers) and unified *dimension object* storing the information of the axes classes.

In Psi4, a perfect candidate for the above is `psi4::Tensor` class declared in `psi4/libthce/thce.h`. Except from the numeric content its instances also store the information of the dimensions in a form of a vector of `psi4::Dimension` instances.

Another possibility is to use `psi::Matrix` objects, instead of `psi4::Tensor` objects, possibly putting them into a `std::vector` container in case there is more than two axes.

Chapter 4

List of One-Electron Potentials

Here I provide the list of OEP's that have been already derived within the scope of the OEPDev project.

4.1 Electrostatic Energy OEP's

For electrostatic energy calculations, OEP is simply the electrostatic potential due to nuclei and electrons.

3D form:

$$v(\mathbf{r}) = \sum_x \frac{Z_x}{|\mathbf{r} - \mathbf{r}_x|} + \sum_{\mu\nu \in A} P_{\nu\mu} \int d\mathbf{r}' \frac{\phi_\mu^*(\mathbf{r}')\phi_\nu(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|}$$

Matrix form:

$$v_{ik} = \sum_{x \in A} Z_x V_{ik}^{(x)} + \sum_{\mu\nu \in A} P_{\nu\mu} (\mu\nu|ik)$$

4.2 Pauli Repulsion OEP's

The following potentials are derived for the evaluation of the Pauli repulsion energy based on Murrell's expressions.

4.2.1 First-order contribution in overlap matrix expansion.

This contribution is simply the electrostatic potential coming from all nuclei and electron density except* from electron density from molecular orbital i that interacts with the generalized overlap density between i of molecule A and j of molecule B .

3D forms:

$$v(\mathbf{r})_{S^{-1}}^{A[i]} = - \sum_{x \in A} \frac{Z_x}{|\mathbf{r} - \mathbf{r}_x|} + \sum_{\mu\nu \in A} \{D_{\nu\mu} - C_{\mu i}^* C_{\nu i}\} \int d\mathbf{r}' \frac{\phi_\mu^*(\mathbf{r}')\phi_\nu(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|}$$

Matrix forms:

$$v_{\xi i}(S^{-1}) = \sum_{\kappa \in A} C_{i\kappa} \left\{ - \sum_{x \in A} V_{\kappa \xi}^{(x)} + \sum_{\mu \nu \in A} \{ D_{\nu \mu} - C_{\mu i}^* C_{\nu i} \} (\mu \nu | \xi \kappa) \right\}$$

4.2.2 Second-order contribution in overlap matrix expansion.

To be added here!

4.3 Excitonic Energy Transfer OEP's

The following potentials are derived for the evaluation of the short-range EET couplings based on Fujimoto's TDFI-TI method.

4.3.1 ET contributions.

3D forms:

$$\begin{aligned} v(\mathbf{r})_1^{A[\mu]} &= -C_{\mu L}^* \sum_{x \in A} \frac{Z_x}{|\mathbf{r} - \mathbf{r}_x|} + \sum_{\nu \kappa \in A} \left\{ C_{\mu L}^* D_{\nu \kappa} - \frac{1}{2} C_{\nu L}^* D_{\mu \kappa} \right\} \int d\mathbf{r}' \frac{\phi_{\nu}^*(\mathbf{r}') \phi_{\kappa}(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} \\ v(\mathbf{r})_2^{A[\mu]} &= C_{\kappa H} \sum_{\nu \kappa \in A} \left\{ 2C_{\nu L}^* C_{\mu H}^* - C_{\nu H}^* C_{\mu L}^* \right\} \int d\mathbf{r}' \frac{\phi_{\nu}^*(\mathbf{r}') \phi_{\kappa}(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} \\ v(\mathbf{r})_3^{A[\mu]} &= v(\mathbf{r})_1^{A[\mu]} + v(\mathbf{r})_2^{A[\mu]} \end{aligned}$$

Matrix forms:

$$\begin{aligned} v_{\mu \xi}(1) &= -C_{\mu L}^* \sum_{x \in A} V_{\mu \xi}^x + \sum_{\nu \kappa \in A} \left\{ C_{\mu L}^* D_{\nu \kappa} - \frac{1}{2} C_{\nu L}^* D_{\mu \kappa} \right\} (\nu \kappa | \mu \xi) \\ v_{\mu \xi}(2) &= C_{\kappa H} \sum_{\nu \kappa \in A} \left\{ 2C_{\nu L}^* C_{\mu H}^* - C_{\nu H}^* C_{\mu L}^* \right\} (\nu \kappa | \mu \xi) \\ v_{\mu \xi}(3) &= v_{\mu \xi}(1) + v_{\mu \xi}(2) \end{aligned}$$

4.3.2 HT contributions.

Do be derived.

4.3.3 CT contributions.

To be derived.

4.4 Full HF Interaction OEP's

The following potentials are derived for the evaluation of the full Hartree-Fock interaction energy based on the OEPDev equations.

Chapter 5

Density-fitting specialized for OEP's

To get the ab-initio representation of a OEP, one can use a procedure similar to the typical density fitting or resolution of identity, both of which are nowadays widely used to compute electron-repulsion integrals (ERI's) more efficiently.

An arbitrary one-electron potential of molecule A acting on any state vector associated with molecule A can be expanded in the *auxiliary space* centered on A as

$$v|i) = \sum_{\xi} v|\xi)(\xi|i)$$

under the necessary assumption that the auxiliary basis set is *complete*. In that case, formally one can write the following identity

$$(\eta|v|i) = \sum_{\xi} (\eta|v|\xi)S_{\xi i}$$

The matrix elements of the OEP operator in auxiliary space can be computed in the same way as the matrix elements with any other basis function. In reality, it is almost impossible to reach the completeness of the basis set, however, but it is possible to obtain the **effective** matrix elements of the OEP operator in auxiliary space, rather than compute them as they are in the above equation explicitly. We expand the LHS of the first equation on this page in a series of the auxiliary basis functions scaled by the undetermined expansion coefficients:

$$v|i) = \sum_{\xi} G_{i\xi}|\xi)$$

The expansion coefficients are the effective matrix elements of the OEP operator in auxiliary basis set. Now, multiplying both sides by another auxiliary basis function and subsequently inverting the equation one obtains the expansion coefficients:

$$G_{i\eta} = \sum_{\xi} (i|v|\eta) [S^{-1}]_{\eta\xi}$$

In this way, it is possible to approximately determine the matrix elements of the OEP operator with any other basis function in case the auxiliary basis set is not complete. **In particular**, when the other basis function does not belong to molecule A but to the (changing) environment, it is straightforward to compute the resulting matrix element, which is simply given as

$$(j_{\in B}|v^A|i_{\in A}) = \sum_{\xi} S_{j\xi} G_{i\xi}$$

where j denotes the other (environmental) basis function.

In the above equation, the OEP-part (fragment parameters for molecule A only) and the Solver-part (subject to be computed by solver on the fly) are separated. This then forms a basis for fragment-based approach to solve Quantum Chemistry problems related to the extended molecular aggregates.

Chapter 6

Implemented Models

6.1 Target Properties

Detailed list of models which is to be implemented in the OEPDev project is given below:

Table 1. Models subject to be implemented and analyzed within oep-dev.

Pauli energy	Induction energy	EET Coupling
EFP2-Pauli	EFP2-Induced Dipoles	TrCAMM
Murrel et al.'s theory	Density Susceptibility	OEP-ET/HT
OEP-Murrel et al.'s		TDFI-TI
		FED
Exact (Stone's)	Exact (incl. CT)	Exact (ESD)

6.2 Target, Benchmark and Competing Models

The target models introduced in the Project shall be tested against the following benchmarks and compared with the following state-of-the-art models:

Table 2. Target models vs benchmarks and competitor models.

Target Model	Benchmarks	Competing Model
OEP-Murrel et al.'s	Murrel et al.'s	EFP2-Pauli
	Exact (Stone's)	
OEP-ET/HT + TrCAMM	Exact (ESD)	TDFI-TI
	FED	FED
	TDFI-TI	
Density Susceptibility	Exact (incl. CT)	EFP2-Induced Dipoles

Chapter 7

Contributing to oep-dev

OepDev is a plugin to Psi4.

Therefore it should follow the programming etiquette of Psi4. Also, oep-dev has additional programming tips to make the code more versatile and easy in further development. Here, I emphasise on most important aspects regarding the **programming rules**.

7.1 Main routine and libraries

Oep-dev has only *one* source file in the plugin base directory, i.e., `main.cc`. This is the main driver routine that handles the functionality of the whole OEP testing platform: specifies options for Psi4 input file and implements test routines based on the options. Other sources are stored in `MODULE/libNAME*` directories where `NAME` is the name of the library with sources and header files, whereas `MODULE` is the directory of the oep-dev module.

Things to remember:

1. **No other sources in base directory.** It is not permitted to place any new source or other files in the plugin base directory (i.e., where `main.cc` resides).
2. **Sources in library directories.** Any additional source code has to be placed in `oepdev/libNAME*` directory (either existing one or a new one; in the latter case remember to add the new `*.cc` files to `CMakeLists.txt` in the plugin base directory).
3. **Miscellanea in special directories.** If you want to add additional documentation, put it in the `doc` directory. If you want to add graphics, put it in the `images` directory.

7.2 Header files in libraries

Header files are handy in obtaining a quick glimpse of the functionality within certain library. Each library directory should contain at least one header file in oep-dev. However, header files can be problematic if not managed properly.

Things to remember:

1. **Header preprocessor variable.** Define the preprocessor variable specifying the existence of include of the particular header file. The format of such is

```
#ifndef MODULE_LIBRARY_HEADER_h
#define MODULE_LIBRARY_HEADER_h
// rest of your code goes here
#endif // MODULE_LIBRARY_HEADER_h
```

Last line is the **end** of the header file. The preprocessor variables represents the directory tree `oepdev/-MODULE/LIBRARY/HEADER.h` structure (where `oepdev` is the base plugin directory). `MODULE` is the

plugin module name (e.g. `oepdev`, the name of the module directory) `LIBRARY` is the name of the library (e.g. `libutil`, should be the same as library directory name) `HEADER` is the name of the header in library directory (e.g. `diis` for `diis.h` header file)

2. **Set module namespace.** To prevent naming clashes with other modules and with `Psi4` it is important to operate in separate namespace (e.g. for a module).

```
namespace MODULE {
// your code goes here
} // EndNameSpace MODULE
```

For instance, all classes and functions in `oepdev` module are implemented within the namespace of the same label. Considering addition of other local namespaces within a module can also be useful in certain cases.

7.3 Environmental variables

Defining the set of intrinsic environmental variables can help in code management and conditional compilation. The `oep-dev` environmental variables are defined in `include/oepdev_files.h` file. Remember also about `psi4` environmental variables defined in `psi4/psifiles.h` header. As a rule, the `oep-dev` environmental variable should have the following format:

```
OEPDEV_XXXX
```

where `XXXX` is the descriptive name of variable.

7.4 Documenting the code

Code has to be documented (at best at a time it is being created). The place for documentation is always in header files. Additional documentation can be also placed in source files. Leaving a chunk of code for a production run without documentation is unacceptable.

Use Doxygen style for documentation all the time. Remember that it supports markdown which can make the documentation even more clear and easy to understand. Additionally you can create a nice `.rst` documentation file for Sphinx program. If you are coding equations, always include formulae in the documentation!

Things to remember:

1. **Descriptions of classes, structures, global functions, etc.** Each programming object should have a description.
2. **Documentation for function arguments and return object.** Usage of functions and class methods should be explained by providing the description of all arguments (use `\param` and `\return` Doxygen keywords).
3. **One-line description of class member variables.** Any class member variable should be preceded by a one-liner documentation (starting from `///`).
4. **Do not be afraid of long names in the code.** Self-documenting code is a blessing!

7.5 Naming conventions

Naming is important because it helps to create more readable and clear self-documented code.

Some loose suggestions:

1. **Do not be afraid of long names in the code, but avoid redundancy.** Examples of good and bad names: good name: `get_density_matrix`; bad name: `get_matrix`. Unless there is only one type of matrix a particular objects can store, `matrix` is not a good name for a getter method. good name: `class Wavefunction`, bad name: `class WFN` good name: `int numberOfErrorVectors`, bad name: `int nvec`, bad name: `the_number_of_error_vectors` good name: `class EFPotential`, probably bad name: `class EffectiveFragmentPotential`. The latter might be understood by some people as a class that inherits from `EffectiveFragment` class. If it is not the case, compromise between abbreviation and long description is OK.
2. **Short names are OK in special situations.** In cases meaning of a particular variable is obvious and it is frequently used in the code locally, it can be named shortly. Examples are: `i` when iterating `no` number of occupied orbitals, `nv` number of virtual orbitals, etc.
3. **Clumped names for variables and dashed names for functions.** Try to distinguish between variable name like `sizeofOEPTypelist` and a method name `get_matrix()` (neither `size_of_OEP_type_list`, nor `getMatrix()`). This is little bit cosmetics, but helps in managing the code when it grows.
4. **Class names start from capital letter.** However, avoid only capital letters in class names, unless it is obvious. Avoid also dashes in class names (they are reserved for global functions and class methods). Examples: good name: `DIISManager`, bad name: `DIIS`. good name: `EETCouplingSolver`, bad name: `EETSolver`, very bad: `EET`.

7.6 Use Object-Oriented Programming

Try to organise your creations in objects having special relationships and data structures. Encapsulation helps in producing self-maintaining code and is much easier to use. Use:

- **factory design** for creating objects
- **container design** for designing data structures
- **polymorphism** when dealing with various flavours of one particular feature in the data structure

Note: In Psi4, factories are frequently implemented as static methods of the base classes, for example `psi::BasisSet::build` static method. It can be followed when building object factories in oep-dev too.

Chapter 8

Namespace Index

8.1 Namespace List

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

oepdev	Oepdev module namespace	??
psi	Psi4 package namespace	??

Chapter 9

Hierarchical Index

9.1 Class Hierarchy

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

oepdev::AllAOIntegralsIterator	??
oepdev::AllAOShellCombinationsIterator	??
oepdev::CPHF	??
CubicScalarGrid	
oepdev::CubePointsCollection3D	??
oepdev::DIISManager	??
enable_shared_from_this	
oepdev::OEPDevSolver	??
oepdev::ElectrostaticEnergySolver	??
oepdev::RepulsionEnergySolver	??
oepdev::OEPotential	??
oepdev::EETCouplingOEPotential	??
oepdev::ElectrostaticEnergyOEPotential	??
oepdev::RepulsionEnergyOEPotential	??
oepdev::ESPSolver	??
oepdev::Points3DIterator::Point	??
oepdev::Points3DIterator	??
oepdev::CubePoints3DIterator	??
oepdev::RandomPoints3DIterator	??
oepdev::PointsCollection3D	??
oepdev::CubePointsCollection3D	??
oepdev::RandomPointsCollection3D	??
PotentialInt	
oepdev::PotentialInt	??
oepdev::ScalarField3D	??
oepdev::ElectrostaticPotential3D	??
oepdev::OEPotential3D< T >	??
Wavefunction	
oepdev::WavefunctionUnion	??

Chapter 10

Class Index

10.1 Class List

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

oepdev::AllAOIntegralsIterator	Loop over all possible ERI within a particular shell	??
oepdev::AllAOShellCombinationsIterator	Loop over all possible ERI shells	??
oepdev::CPHF	CPHF solver class	??
oepdev::CubePoints3DIterator	Iterator over a collection of points in 3D space. g09 Cube-like order	??
oepdev::CubePointsCollection3D	G09 cube-like ordered collection of points in 3D space	??
oepdev::DIISManager	DIIS manager	??
oepdev::EETCouplingOEPotential	Generalized One-Electron Potential for EET coupling calculations	??
oepdev::ElectrostaticEnergyOEPotential	Generalized One-Electron Potential for Electrostatic Energy calculations	??
oepdev::ElectrostaticEnergySolver	Compute the Coulombic interaction energy between unperturbed wavefunctions	??
oepdev::ElectrostaticPotential3D	Electrostatic potential of a molecule	??
oepdev::ESPSolver	Charges from Electrostatic Potential (ESP). A solver-type class	??
oepdev::OEPDevSolver	Solver of properties of molecular aggregates. Abstract base	??
oepdev::OEPotential	Generalized One-Electron Potential: Abstract base	??
oepdev::OEPotential3D< T >	Class template for OEP scalar fields	??
oepdev::Points3DIterator::Point	??
oepdev::Points3DIterator	Iterator over a collection of points in 3D space. Abstract base	??
oepdev::PointsCollection3D	Collection of points in 3D space. Abstract base	??
oepdev::PotentialInt	Computes potential integrals	??
oepdev::RandomPoints3DIterator	Iterator over a collection of points in 3D space. Random collection	??

oepdev::RandomPointsCollection3D	
Collection of random points in 3D space	??
oepdev::RepulsionEnergyOEPotential	
Generalized One-Electron Potential for Pauli repulsion energy calculations	??
oepdev::RepulsionEnergySolver	
Compute the Pauli-Repulsion interaction energy between unperturbed wavefunctions	??
oepdev::ScalarField3D	
Scalar field in 3D space. Abstract base	??
oepdev::WavefunctionUnion	
Union of two Wavefunction objects	??

Chapter 11

Namespace Documentation

11.1 oepdev Namespace Reference

oepdev module namespace.

Classes

- class [OEPotential](#)
Generalized One-Electron Potential: Abstract base.
- class [ElectrostaticEnergyOEPotential](#)
Generalized One-Electron Potential for Electrostatic Energy calculations.
- class [RepulsionEnergyOEPotential](#)
Generalized One-Electron Potential for Pauli repulsion energy calculations.
- class [EETCouplingOEPotential](#)
Generalized One-Electron Potential for EET coupling calculations.
- class [PotentialInt](#)
Computes potential integrals.
- class [CPHF](#)
CPHF solver class.
- class [DIISManager](#)
DIIS manager.
- class [ESPSolver](#)
Charges from Electrostatic Potential (ESP). A solver-type class.
- class [AllAOShellCombinationsIterator](#)
Loop over all possible ERI shells.
- class [AllAOIntegralsIterator](#)
Loop over all possible ERI within a particular shell.
- class [OEPDevSolver](#)
Solver of properties of molecular aggregates. Abstract base.
- class [ElectrostaticEnergySolver](#)
Compute the Coulombic interaction energy between unperturbed wavefunctions.
- class [RepulsionEnergySolver](#)
Compute the Pauli-Repulsion interaction energy between unperturbed wavefunctions.
- class [Points3DIterator](#)
Iterator over a collection of points in 3D space. Abstract base.
- class [CubePoints3DIterator](#)
Iterator over a collection of points in 3D space. g09 Cube-like order.

- class [RandomPoints3DIterator](#)
Iterator over a collection of points in 3D space. Random collection.
- class [PointsCollection3D](#)
Collection of points in 3D space. Abstract base.
- class [RandomPointsCollection3D](#)
Collection of random points in 3D space.
- class [CubePointsCollection3D](#)
G09 cube-like ordered collection of points in 3D space.
- class [ScalarField3D](#)
Scalar field in 3D space. Abstract base.
- class [ElectrostaticPotential3D](#)
Electrostatic potential of a molecule.
- class [OEPotential3D](#)
Class template for OEP scalar fields.
- class [WavefunctionUnion](#)
Union of two Wavefunction objects.

Typedefs

- using **SharedWavefunction** = std::shared_ptr< Wavefunction >
- using **SharedBasisSet** = std::shared_ptr< BasisSet >
- using **SharedTensor** = std::shared_ptr< Tensor >
- using **SharedMatrix** = std::shared_ptr< Matrix >
- using **SharedVector** = std::shared_ptr< Vector >
- using **SharedScalarField3D** = std::shared_ptr< [ScalarField3D](#) >
- using **SharedIntegralFactory** = std::shared_ptr< IntegralFactory >
- using **SharedTwoBodyAOInt** = std::shared_ptr< TwoBodyAOInt >
- using **SharedWavefunctionUnion** = std::shared_ptr< [WavefunctionUnion](#) >
- using **SharedOEPotential** = std::shared_ptr< [OEPotential](#) >
- using **SharedMolecule** = std::shared_ptr< Molecule >
- using **SharedSuperFunctional** = std::shared_ptr< SuperFunctional >
- using **SharedMOSpace** = std::shared_ptr< MOSpace >
- using **SharedMOSpaceVector** = std::vector< std::shared_ptr< MOSpace >>
- using **SharedIntegralTransform** = std::shared_ptr< IntegralTransform >
- using **SharedLocalizer** = std::shared_ptr< Localizer >

Functions

- void [preamble](#) (void)
Print preamble for module OEPDEV.
- std::shared_ptr< SuperFunctional > [create_superfunctional](#) (std::string name, Options &options)
Set up DFT functional.
- std::shared_ptr< Molecule > [extract_monomer](#) (std::shared_ptr< const Molecule > molecule_dimer, int id)
Extract molecule from dimer.
- std::shared_ptr< Wavefunction > [solve_scf](#) (std::shared_ptr< Molecule > molecule, std::shared_ptr< BasisSet > primary, std::shared_ptr< SuperFunctional > functional, Options &options, std::shared_ptr< PSIO > psio)
Solve RHF-SCF equations for a given molecule in a given basis set.

11.1.1 Detailed Description

oepdev module namespace. Contains:

11.1.2 Function Documentation

11.1.2.1 `std::shared_ptr< SuperFunctional > oepdev::create_superfunctional (std::string name, Options & options)`

Set up DFT functional.

Now it accepts only pure HF functional.

Parameters

<i>name</i>	name of the functional ("HF" is now only available)
<i>options</i>	psi::Options object

Returns

psi::SharedSuperFunctional object with functional.

11.1.2.2 `std::shared_ptr< Molecule > oepdev::extract_monomer (std::shared_ptr< const Molecule > molecule_dimer, int id)`

Extract molecule from dimer.

Parameters

<i>molecule_dimer</i>	psi::SharedMolecule object with dimer
<i>id</i>	index of a molecule (starts from 1)

Returns

psi::SharedMolecule object with indicated monomer

11.1.2.3 `std::shared_ptr< Wavefunction > oepdev::solve_scf (std::shared_ptr< Molecule > molecule, std::shared_ptr< BasisSet > primary, std::shared_ptr< SuperFunctional > functional, Options & options, std::shared_ptr< PSIO > psio)`

Solve RHF-SCF equations for a given molecule in a given basis set.

Parameters

<i>molecule</i>	psi::SharedMolecule object with molecule
<i>primary</i>	shared primary basis set
<i>functional</i>	DFT functional
<i>options</i>	psi::Options object
<i>psio</i>	psi::PSIO object

Returns

psi::SharedWavefunction SCF wavefunction of the molecule

11.2 psi Namespace Reference

Psi4 package namespace.

Typedefs

- using **SharedVetor** = std::shared_ptr< Vector >

- using **SharedBasisSet** = std::shared_ptr< BasisSet >
- using **SharedMolecule** = std::shared_ptr< Molecule >
- using **SharedMatrix** = std::shared_ptr< Matrix >
- using **SharedWavefunction** = std::shared_ptr< Wavefunction >

Functions

- int [read_options](#) (std::string name, Options &options)
Options for the OEPDEV plugin.
- SharedWavefunction [oepdev](#) (SharedWavefunction ref_wfn, Options &options)
Main routine of the OEPDEV plugin.

11.2.1 Detailed Description

Psi4 package namespace. Contains all Psi4 functionalities.

11.2.2 Function Documentation

11.2.2.1 SharedWavefunction psi::oepdev (SharedWavefunction ref_wfn, Options & options)

Main routine of the OEPDEV plugin.

Created with intention to test various models of the interaction energy between two molecules, described by the Hartree-Fock-Roothaan-Hall theory or the configuration interaction with singles theory.

In particular, the plugin tests the models of:

1. the Pauli exchange-repulsion interaction energy (Project II)
2. the Induction interaction energy (Project III)
3. the excitation energy transfer couplings (Project I)

against benchmarks (exact or reference solutions). Detailed list of models is given below:

		Interaction Property	
Pauli energy		Induction energy	EET Coupling
Methods			
EFP2-Pauli	EFP2-Induced Dipoles	TrCAMM	
Murrel et al.'s theory	Density Susceptibility	OEP-ET/HT	
OEP-Murrel et al.'s	TDFI-TI		FED
	Exact (Stone's)	Exact (incl. CT)	Exact (ESD)

The target models introduced in the Project shall be tested against the following benchmarks and compared with the following state-of-the-art models:

Target Model	Benchmarks	Competing Model
Murrel et al.'s	Murrel et al.'s	EFP2-Pauli
OEP-ET/HT	TrCAMM	Exact (Stone's)
Density Susceptibility	Exact (ESD)	TDFI-TI
		FED
		TDFI-TI
		Exact (incl. CT)
		EFP2-Induced Dipoles

Parameters of the plugin driver:

Parameters

<i>ref_wfn</i>	shared wavefunction of a dimer
<i>options</i>	psi::Options object

Returns

psi::SharedWavefunction (as for now the same as ref_wfn)

11.2.2.2 int psi::read_options (std::string *name*, Options & *options*)

Options for the OEPDEV plugin.

Parameters

<i>name</i>	name of driver function
<i>options</i>	psi::Options object

Returns

true

Chapter 12

Class Documentation

12.1 oepdev::AllAOIntegralsIterator Class Reference

Loop over all possible ERI within a particular shell.

```
#include <integrals_iter.h>
```

Public Member Functions

- [AllAOIntegralsIterator](#) (const [AllAOShellCombinationsIterator](#) &shellIter)
Construct by shell iterator (const object)
- [AllAOIntegralsIterator](#) (std::shared_ptr< [AllAOShellCombinationsIterator](#) > shellIter)
Construct by shell iterator (pointed by shared pointer)
- void [first](#) ()
First iteration.
- void [next](#) ()
Next iteration.
- bool [is_done](#) ()
Check status of iterations.
- int [i](#) () const
Grab the current integral i index.
- int [j](#) () const
Grab the current integral j index.
- int [k](#) () const
Grab the current integral k index.
- int [l](#) () const
Grab the current integral l index.
- int [index](#) () const

12.1.1 Detailed Description

Loop over all possible ERI within a particular shell.

Constructed by providing a const reference or shared pointer to an [AllAOShellCombinationsIterator](#) object.

Suggested usage:

See Also

[AllAOShellCombinationsIterator](#)

12.1.2 Constructor & Destructor Documentation

12.1.2.1 oepdev::AllAOIntegralsIterator::AllAOIntegralsIterator (const AllAOShellCombinationsIterator & shellIter)

Construct by shell iterator (const object)

Parameters

<i>shellIter</i>	- shell iterator object
------------------	-------------------------

12.1.2.2 oepdev::AllAOIntegralsIterator::AllAOIntegralsIterator (std::shared_ptr< AllAOShellCombinationsIterator > shellIter)

Construct by shell iterator (pointed by shared pointer)

Parameters

<i>shellIter</i>	- shell iterator object
------------------	-------------------------

12.1.3 Member Function Documentation

12.1.3.1 int oepdev::AllAOIntegralsIterator::index () const [inline]

Grab the current index of integral value stored in the buffer

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

- oepdev/libutil/integrals_iter.h
- oepdev/libutil/integrals_iter.cc

12.2 oepdev::AllAOShellCombinationsIterator Class Reference

Loop over all possible ERI shells.

```
#include <integrals_iter.h>
```

Public Member Functions

- [AllAOShellCombinationsIterator](#) (SharedBasisSet [bs_1](#), SharedBasisSet [bs_2](#), SharedBasisSet [bs_3](#), SharedBasisSet [bs_4](#))
Construct by providing basis sets for each axis. The basis sets must be defined for the same molecule.
- [AllAOShellCombinationsIterator](#) (SharedIntegralFactory integrals)
Construct by providing integral factory.
- void [first](#) ()
First iteration.
- void [next](#) ()
Next iteration.
- bool [is_done](#) ()
Check status of iterations.
- int [P](#) () const
Grab the current shell P index.
- int [Q](#) () const
Grab the current shell Q index.

- int **R** () const
Grab the current shell R index.
- int **S** () const
Grab the current shell S index.
- SharedBasisSet **bs_1** () const
Grab the basis set of axis 1.
- SharedBasisSet **bs_2** () const
Grab the basis set of axis 2.
- SharedBasisSet **bs_3** () const
Grab the basis set of axis 3.
- SharedBasisSet **bs_4** () const
Grab the basis set of axis 4.
- void **compute_shell** (SharedTwoBodyAOInt tei) const
Compute ERI's for the current shell. The eris are stored in the buffer of the argument object.

12.2.1 Detailed Description

Loop over all possible ERI shells.

Constructed by providing shared pointer to IntegralFactory object or shared pointers to four basis set spaces.

Suggested usage:

```
SharedIntegralFactory ints = std::make_shared<IntegralFactory>(bs1, bs2, bs3, bs4);
SharedTwoBodyAOInt tei(ints->eri());
AllAOShellCombinationsIterator shellIter(ints);
const double * buffer = tei->buffer();
for (shellIter.first(); shellIter.is_done()==false; shellIter.next())
{
    shellIter.compute_shell(tei);
    AllAOIntegralsIterator intsIter(shellIter);
    for (intsIter.first(); intsIter.is_done()==false; intsIter.next())
    {
        // Grab (ij|kl) integrals and indices here
        int i = intsIter.i();
        int j = intsIter.j();
        int k = intsIter.k();
        int l = intsIter.l();
        double integral = buffer[intsIter.index()];
    }
}
```

12.2.2 Constructor & Destructor Documentation

12.2.2.1 oepdev::AllAOShellCombinationsIterator::AllAOShellCombinationsIterator (SharedBasisSet *bs_1*, SharedBasisSet *bs_2*, SharedBasisSet *bs_3*, SharedBasisSet *bs_4*)

Construct by providing basis sets for each axis. The basis sets must be defined for the same molecule.

Parameters

<i>bs_1</i>	- basis set of axis 1
<i>bs_2</i>	- basis set of axis 2
<i>bs_3</i>	- basis set of axis 3
<i>bs_4</i>	- basis set of axis 4

12.2.2.2 oepdev::AllAOShellCombinationsIterator::AllAOShellCombinationsIterator (SharedIntegralFactory *integrals*)

Construct by providing integral factory.

Parameters

<i>integrals</i>	- integral factory object
------------------	---------------------------

12.2.3 Member Function Documentation

12.2.3.1 void oepdev::AllAOShellCombinationsIterator::compute_shell (SharedTwoBodyAOInt *tei*) const

Compute ERI's for the current shell. The eris are stored in the buffer of the argument object.

Parameters

<i>tei</i>	- two electron AO integral
------------	----------------------------

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

- oepdev/libutil/integrals_iter.h
- oepdev/libutil/integrals_iter.cc

12.3 oepdev::CPHF Class Reference

[CPHF](#) solver class.

```
#include <cphf.h>
```

Public Member Functions

- [CPHF](#) (SharedWavefunction ref_wfn, Options &options)
orbital-associated polarizabilities tensors
- [~CPHF](#) ()
Destructor.
- void [compute](#) (void)
run the calculations
- void [print](#) (void) const
print to output file
- std::shared_ptr< Matrix > [get_molecular_polarizability](#) (void) const
retrieve the molecular (total) polarizability

Protected Attributes

- const int [_no](#)
Number of occupied orbitals.
- const int [_nv](#)
Number of virtual orbitals.
- const int [_nn](#)
Number of basis functions.
- long int [_memory](#)
Memory.
- int [_maxiter](#)
Maximum number of iterations.
- double [_conv](#)
[CPHF](#) convergence threshold.

- bool [_with_diis](#)
whether use DIIS or not
- const int [_diis_dim](#)
Size of subspace.
- std::shared_ptr< BasisSet > [_primary](#)
Primary Basis Set.
- std::shared_ptr< Matrix > [_cocc](#)
Occupied orbitals.
- std::shared_ptr< Matrix > [_cvir](#)
Virtual orbitals.
- std::shared_ptr< Vector > [_eps_occ](#)
Occupied orbital energies.
- std::shared_ptr< Vector > [_eps_vir](#)
Virtual orbital energies.
- std::vector< std::shared_ptr
< [oepdev::DIISManager](#) > > [_diis](#)
the DIIS managers for each perturbation operator x, y and z
- Options [_options](#)
Options.
- std::shared_ptr< Matrix > [_molecular_polarizability](#)
Total (molecular) polarizability tensor.

12.3.1 Detailed Description

[CPHF](#) solver class.

Solves [CPHF](#) equations (now only for RHF wavefunction). Computes molecular and orbital-associated polarizabilities.

Suggested usage:

```
std::shared_ptr<CPHF> cphf(new CPHF(ref_wfn, options)); cphf->compute(); std::shared_ptr<Matrix> polar-
izability = cphf->get_molecular_polarizability(); std::shared_ptr<Tensor> orbital_polars = cphf->get_orbital_-
polarizabilities();
```

12.3.2 Constructor & Destructor Documentation

12.3.2.1 oepdev::CPHF::CPHF (SharedWavefunction ref_wfn, Options & options)

orbital-associated polarizabilities tensors

Constructor

Parameters

<i>ref_wfn</i>	reference HF wavefunction
<i>options</i>	set of Psi4 options

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

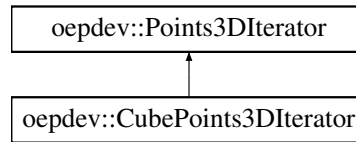
- oepdev/libutil/cphf.h
- oepdev/libutil/cphf.cc

12.4 oepdev::CubePoints3DIterator Class Reference

Iterator over a collection of points in 3D space. g09 Cube-like order.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::CubePoints3DIterator:



Public Member Functions

- **CubePoints3DIterator** (const int &nx, const int &ny, const int &nz, const double &dx, const double &dy, const double &dz, const double &ox, const double &oy, const double &oz)
- virtual void [first](#) ()
Initialize first iteration.
- virtual void [next](#) ()
Step to next iteration.

Protected Attributes

- const int **nx_**
- const int **ny_**
- const int **nz_**
- const double **dx_**
- const double **dy_**
- const double **dz_**
- const double **ox_**
- const double **oy_**
- const double **oz_**
- int **ii_**
- int **jj_**
- int **kk_**

Additional Inherited Members

12.4.1 Detailed Description

Iterator over a collection of points in 3D space. g09 Cube-like order.

Note: Always create instances by using static factory method from [Points3DIterator](#). Do not use constructor of this class.

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

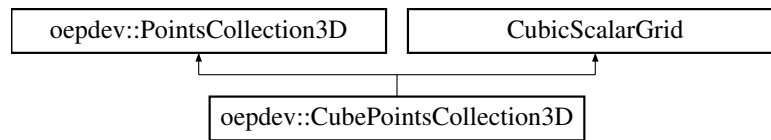
- oepdev/libutil/space3d.h
- oepdev/libutil/space3d.cc

12.5 oepdev::CubePointsCollection3D Class Reference

G09 cube-like ordered collection of points in 3D space.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::CubePointsCollection3D:



Public Member Functions

- **CubePointsCollection3D** ([Collection](#) collectionType, const int &nx, const int &ny, const int &nz, const double &px, const double &py, const double &pz, psi::SharedBasisSet bs, psi::Options &options)
- virtual void [print](#) () const
Print the information to Psi4 output file.
- virtual void **write_cube_file** (psi::SharedMatrix v, const std::string &name)

Additional Inherited Members

12.5.1 Detailed Description

G09 cube-like ordered collection of points in 3D space.

Note: Do not use constructors of this class explicitly. Instead, use static factory methods of the superclass to create instances.

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

- oepdev/libutil/space3d.h
- oepdev/libutil/space3d.cc

12.6 oepdev::DIISManager Class Reference

DIIS manager.

```
#include <diis.h>
```

Public Member Functions

- [DIISManager](#) (int dim, int na, int nb)
- [~DIISManager](#) ()
Destructor.
- void [put](#) (const std::shared_ptr< const Matrix > &error, const std::shared_ptr< const Matrix > &vector)
- void [compute](#) (void)
- void [update](#) (std::shared_ptr< Matrix > &other)

12.6.1 Detailed Description

DIIS manager.

Instance can interact directly with the process of solving vector quantities in iterative manner. One needs to pass the dimensions of solution vector as well as the DIIS subspace size. The iterative procedure requires providing the current vector and also an estimate of the error vector. The updated DIIS vector can be copied to an old vector through the Instance.

12.6.2 Constructor & Destructor Documentation

12.6.2.1 oepdev::DIISManager::DIISManager (int *dim*, int *na*, int *nb*)

Constructor.

Parameters

<i>dim</i>	Size of DIIS subspace
<i>na</i>	Number of solution rows
<i>nb</i>	Number of solution columns

12.6.3 Member Function Documentation

12.6.3.1 void oepdev::DIISManager::compute (void)

Perform DIIS interpolation.

12.6.3.2 void oepdev::DIISManager::put (const std::shared_ptr< const Matrix > & *error*, const std::shared_ptr< const Matrix > & *vector*)

Put the current solution to the DIIS manager.

Parameters

<i>error</i>	Shared matrix with current solution error
<i>vector</i>	Shared matrix with current solution vector

12.6.3.3 void oepdev::DIISManager::update (std::shared_ptr< Matrix > & *other*)

Update solution vector. Pass the Shared pointer to current solution. Then it will be overridden by the updated DIIS solution.

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

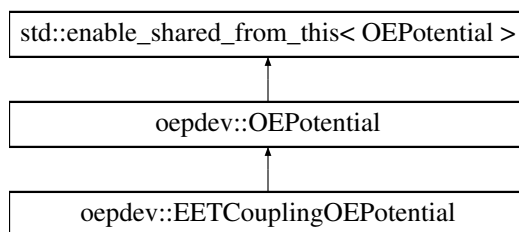
- oepdev/libutil/diis.h
- oepdev/libutil/diis.cc

12.7 oepdev::EETCouplingOEPotential Class Reference

Generalized One-Electron Potential for EET coupling calculations.

```
#include <oep.h>
```

Inheritance diagram for oepdev::EETCouplingOEPotential:



Public Member Functions

- **EETCouplingOEPotential** (SharedWavefunction [wfn](#), SharedBasisSet auxiliary, Options &options)
- **EETCouplingOEPotential** (SharedWavefunction [wfn](#), Options &options)
- virtual void **compute** (const std::string &oepType) override
- virtual void [compute_3D](#) (const std::string &oepType, const double &x, const double &y, const double &z, double &v) override
- virtual void **print_header** () const override

Additional Inherited Members

12.7.1 Detailed Description

Generalized One-Electron Potential for EET coupling calculations.

Contains the following OEP types: "ET1" "ET2" "HT1" "HT1" "HT2" "CT1" "CT2"

12.7.2 Member Function Documentation

12.7.2.1 void EETCouplingOEPotential::compute_3D (const std::string & oepType, const double & x, const double & y, const double & z, double & v) `[override], [virtual]`

Compute value of potential in point x, y, z and save at v

Implements [oepdev::OEPotential](#).

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

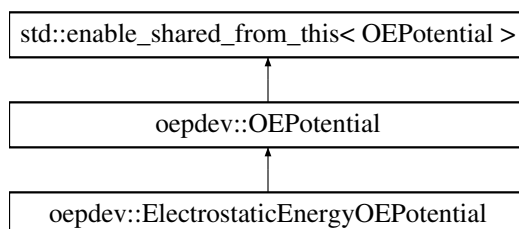
- oepdev/liboep/oep.h
- oepdev/liboep/oep.cc

12.8 oepdev::ElectrostaticEnergyOEPotential Class Reference

Generalized One-Electron Potential for Electrostatic Energy calculations.

```
#include <oep.h>
```

Inheritance diagram for oepdev::ElectrostaticEnergyOEPotential:



Public Member Functions

- [ElectrostaticEnergyOEPotential](#) (SharedWavefunction [wfn](#), Options &options)
Only ESP-based potential is worth implementing.
- virtual void **compute** (const std::string &oepType) override
- virtual void [compute_3D](#) (const std::string &oepType, const double &x, const double &y, const double &z, double &v) override
- virtual void **print_header** () const override

Additional Inherited Members

12.8.1 Detailed Description

Generalized One-Electron Potential for Electrostatic Energy calculations.

Contains the following OEP types: "V"

12.8.2 Member Function Documentation

12.8.2.1 void [ElectrostaticEnergyOEPotential::compute_3D](#) (const std::string & *oepType*, const double & *x*, const double & *y*, const double & *z*, double & *v*) [\[override\]](#), [\[virtual\]](#)

Compute value of potential in point x, y, z and save at v

Implements [oepdev::OEPotential](#).

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

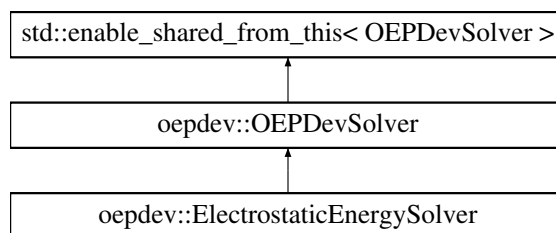
- oepdev/liboep/oep.h
- oepdev/liboep/oep.cc

12.9 oepdev::ElectrostaticEnergySolver Class Reference

Compute the Coulombic interaction energy between unperturbed wavefunctions.

```
#include <solver.h>
```

Inheritance diagram for oepdev::ElectrostaticEnergySolver:



Public Member Functions

- **ElectrostaticEnergySolver** (SharedWavefunctionUnion wfn_union)
- virtual double [compute_oep_based](#) (const std::string &method="DEFAULT")
Compute property by using OEP's.
- virtual double [compute_benchmark](#) (const std::string &method="DEFAULT")
Compute property by using benchmark method.

Additional Inherited Members

12.9.1 Detailed Description

Compute the Coulombic interaction energy between unperturbed wavefunctions.

The implemented methods are shown in below

Keyword	Method Description
Benchmark Methods	
AO_EXPANDED	*Default*. Exact Coulombic energy from atomic orbital expansions.
MO_EXPANDED	Exact Coulombic energy from molecular orbital expansions
OEP-Based Methods	
ESP_SYMMETRIZED	*Default*. Coulombic energy from ESP charges interacting with nuclei and electronic density. Symmetrized with respect to monomers.

Table 12.1: Methods available in the Solver

Below the detailed description of the above methods is given.

Benchmark Methods

Exact Coulombic energy from atomic orbital expansions.

The Coulombic interaction energy is given by

$$E^{\text{Coul}} = E^{\text{Nuc-Nuc}} + E^{\text{Nuc-El}} + E^{\text{El-El}}$$

where the nuclear-nuclear repulsion energy is

$$E^{\text{Nuc-Nuc}} = \sum_{x \in A} \sum_{y \in B} \frac{Z_x Z_y}{|\mathbf{r}_x - \mathbf{r}_y|}$$

the nuclear-electronic attraction energy is

$$E^{\text{Nuc-El}} = \sum_{x \in A} \sum_{\lambda \sigma \in B} Z_x V_{\lambda \sigma}^{(x)} \left(D_{\lambda \sigma}^{(\alpha)} + D_{\lambda \sigma}^{(\beta)} \right) + \sum_{y \in B} \sum_{\mu \nu \in A} Z_y V_{\mu \nu}^{(y)} \left(D_{\mu \nu}^{(\alpha)} + D_{\mu \nu}^{(\beta)} \right)$$

and the electron-electron repulsion energy is

$$E^{\text{El-El}} = \sum_{\mu \nu \in A} \sum_{\lambda \sigma \in B} \left\{ D_{\mu \nu}^{(\alpha)} + D_{\mu \nu}^{(\beta)} \right\} \left\{ D_{\lambda \sigma}^{(\alpha)} + D_{\lambda \sigma}^{(\beta)} \right\} (\mu \nu | \lambda \sigma)$$

In the above equations,

$$V_{\lambda \sigma}^{(x)} \equiv \int \frac{\varphi_{\lambda}^*(\mathbf{r}) \varphi_{\sigma}(\mathbf{r})}{|\mathbf{r} - \mathbf{r}_x|} d\mathbf{r}$$

Exact Coulombic energy from molecular orbital expansion.

This approach is fully equivalent to the atomic orbital expansion shown above. For the closed shell case, the Coulombic interaction energy is given by

$$E^{\text{Coul}} = E^{\text{Nuc-Nuc}} + E^{\text{Nuc-El}} + E^{\text{El-El}}$$

where the nuclear-nuclear repulsion energy is

$$E^{\text{Nuc-Nuc}} = \sum_{x \in A} \sum_{y \in B} \frac{Z_x Z_y}{|\mathbf{r}_x - \mathbf{r}_y|}$$

the nuclear-electronic attraction energy is

$$E^{\text{Nuc-EI}} = 2 \sum_{i \in A} \sum_{y \in B} V_{ii}^{(y)} + 2 \sum_{j \in B} \sum_{x \in A} V_{jj}^{(x)}$$

and the electron-electron repulsion energy is

$$E^{\text{EI-EI}} = 4 \sum_{i \in A} \sum_{j \in B} (ii|jj)$$

OEP-Based Methods

Coulombic energy from ESP charges interacting with nuclei and electronic density.

In this approach, nuclear and electronic density of either species is approximated by ESP charges. In order to achieve symmetric expression, the interaction is computed twice (ESP of A interacting with density matrix and nuclear charges of B and vice versa) and then divided by 2. Thus,

$$E^{\text{Coul}} \approx \frac{1}{2} \left[\sum_{x \in A} \sum_{y \in B} \frac{Z_x q_y}{|\mathbf{r}_x - \mathbf{r}_y|} + \sum_{y \in B} \sum_{\mu \nu \in A} q_y V_{\mu \nu}^{(y)} \left(D_{\mu \nu}^{(\alpha)} + D_{\mu \nu}^{(\beta)} \right) + \sum_{y \in B} \sum_{x \in A} \frac{q_x Z_y}{|\mathbf{r}_x - \mathbf{r}_y|} + \sum_{x \in A} \sum_{\lambda \sigma \in B} Z_x V_{\lambda \sigma}^{(x)} \left(D_{\lambda \sigma}^{(\alpha)} + D_{\lambda \sigma}^{(\beta)} \right) \right]$$

If the basis set is large and the number of ESP centres $q_{x(y)}$ is sufficient, the sum of first two contributions equals the sum of the latter two contributions.

Notes:

- This solver also computes and prints the ESP-ESP point charge interaction energy,

$$E^{\text{Coul,ESP}} \approx \sum_{x \in A} \sum_{y \in B} \frac{q_x q_y}{|\mathbf{r}_x - \mathbf{r}_y|}$$

for reference purposes.

- In order to construct this solver, **always** use the `OEPDevSolver::build` static factory method.

12.9.2 Member Function Documentation

12.9.2.1 `double ElectrostaticEnergySolver::compute_benchmark (const std::string & method = "DEFAULT")`
[virtual]

Compute property by using benchmark method.

Each solver object has one `DEFAULT` benchmark method

Parameters

<i>method</i>	- benchmark method
---------------	--------------------

Implements `oepdev::OEPDevSolver`.

12.9.2.2 `double ElectrostaticEnergySolver::compute_oep_based (const std::string & method = "DEFAULT")`
[virtual]

Compute property by using OEP's.

Each solver object has one `DEFAULT` OEP-based method.

Parameters

<i>method</i>	- flavour of OEP model
---------------	------------------------

Implements [oepdev::OEPDevSolver](#).

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

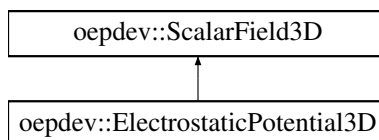
- oepdev/libutil/solver.h
- oepdev/libutil/solver.cc

12.10 oepdev::ElectrostaticPotential3D Class Reference

Electrostatic potential of a molecule.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::ElectrostaticPotential3D:



Public Member Functions

- **ElectrostaticPotential3D** (const int &np, const double &padding, psi::SharedWavefunction [wfn](#), psi::Options &options)
- **ElectrostaticPotential3D** (const int &nx, const int &ny, const int &nz, const double &px, const double &py, const double &pz, psi::SharedWavefunction [wfn](#), psi::Options &options)
- virtual double [compute_xyz](#) (const double &x, const double &y, const double &z)
Compute a value of scalar field at point (x, y, z)
- virtual void [print](#) () const
Print information of the object to Psi4 output.

Additional Inherited Members

12.10.1 Detailed Description

Electrostatic potential of a molecule.

Computes the electrostatic potential of a molecule directly from the wavefunction. The electrostatic potential $v(\mathbf{r})$ at point \mathbf{r} is computed from the following formula:

$$v(\mathbf{r}) = v_{\text{nuc}}(\mathbf{r}) + v_{\text{el}}(\mathbf{r})$$

where the nuclear and electronic contributions are defined accordingly as

$$v_{\text{nuc}}(\mathbf{r}) = \sum_x \frac{Z_x}{|\mathbf{r} - \mathbf{r}_x|}$$

$$v_{\text{el}}(\mathbf{r}) = \sum_{\mu\nu} \left\{ D_{\mu\nu}^{(\alpha)} + D_{\mu\nu}^{(\beta)} \right\} V_{\nu\mu}(\mathbf{r})$$

In the above equations, Z_x denotes the charge of x th nucleus, $D_{\mu\nu}^{(\omega)}$ is the one-particle (relaxed) density matrix element in AO basis associated with the ω electron spin, and $V_{\mu\nu}(\mathbf{r})$ is the potential one-electron integral defined

by

$$V_{v\mu}(\mathbf{r}) \equiv \int d\mathbf{r}' \phi_v^*(\mathbf{r}') \frac{1}{|\mathbf{r} - \mathbf{r}'|} \phi_\mu(\mathbf{r}')$$

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

- oepdev/libutil/space3d.h
- oepdev/libutil/space3d.cc

12.11 oepdev::ESPSolver Class Reference

Charges from Electrostatic Potential (ESP). A solver-type class.

```
#include <esp.h>
```

Public Member Functions

- [ESPSolver](#) (SharedScalarField3D field)
Construct from scalar field.
- [ESPSolver](#) (SharedScalarField3D field, psi::SharedMatrix [centres](#))
Construct from scalar field.
- virtual [~ESPSolver](#) ()
Destructor.
- virtual psi::SharedVector [charges](#) () const
Get the (fit) charges.
- virtual psi::SharedMatrix [centres](#) () const
Get the charge distribution centres.
- virtual void [compute](#) ()
Perform fitting of effective charges.

Protected Attributes

- const int [nCentres_](#)
Number of fit centres.
- SharedScalarField3D [field_](#)
Scalar field.
- psi::SharedVector [charges_](#)
Charges to be fit.
- psi::SharedMatrix [centres_](#)
Centres, at which fit charges will reside.

12.11.1 Detailed Description

Charges from Electrostatic Potential (ESP). A solver-type class.

Solves the least-squares problem to fit the generalized charges q_m , that reproduce the reference generalized potential $v^{\text{ref}}(\mathbf{r})$ supplied by the [ScalarField3D](#) object:

$$\int d\mathbf{r}' \left[v^{\text{ref}}(\mathbf{r}') - \sum_m \frac{q_m}{|\mathbf{r}' - \mathbf{r}_m|} \right]^2 \rightarrow \text{minimize}$$

The charges are subject to the following constraint:

$$\sum_m q_m = 0$$

Method description.

M generalized charges is found by solving the matrix equation

$$\begin{pmatrix} \mathbf{A} & 0 \\ 0 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} \mathbf{b} \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{q} \\ \lambda \end{pmatrix}$$

where the \mathbf{A} matrix of dimension $M \times M$ and \mathbf{b} vector of length M are given as

$$A_{mn} = \sum_i \frac{1}{r_{im} r_{in}}$$

$$b_m = \sum_i \frac{v^{\text{ref}}(\mathbf{r}_m)}{r_{im}}$$

In the above equation, summations run over all sample points, at which reference potential is known.

12.11.2 Constructor & Destructor Documentation

12.11.2.1 oepdev::ESPSolver::ESPSolver (SharedScalarField3D *field*)

Construct from scalar field.

Assume that the centres are on atoms associated with the scalar field.

Parameters

<i>field</i>	- oepdev scalar field object
--------------	------------------------------

12.11.2.2 oepdev::ESPSolver::ESPSolver (SharedScalarField3D *field*, psi::SharedMatrix *centres*)

Construct from scalar field.

Solve ESP equations for a custom set of charge distribution centres.

Parameters

<i>field</i>	- oepdev scalar field object
<i>centres</i>	- matrix with coordinates of charge distribution centres

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

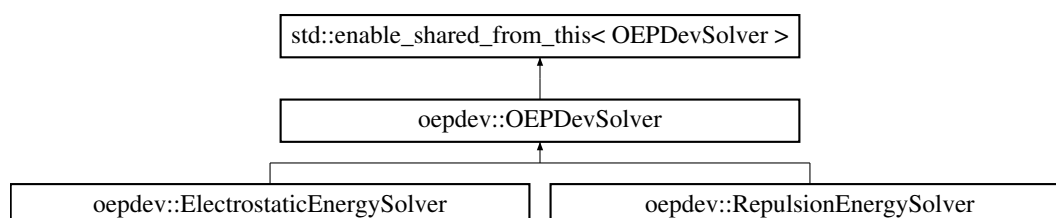
- oepdev/libutil/esp.h
- oepdev/libutil/esp.cc

12.12 oepdev::OEPDevSolver Class Reference

Solver of properties of molecular aggregates. Abstract base.

```
#include <solver.h>
```

Inheritance diagram for oepdev::OEPDevSolver:



Public Member Functions

- [OEPDevSolver](#) (SharedWavefunctionUnion wfn_union)
Take wavefunction union and initialize the Solver.
- virtual [~OEPDevSolver](#) ()
Destructor.
- virtual double [compute_oeplib](#) (const std::string &method="DEFAULT")=0
Compute property by using OEP's.
- virtual double [compute_benchmark](#) (const std::string &method="DEFAULT")=0
Compute property by using benchmark method.

Static Public Member Functions

- static std::shared_ptr
< [OEPDevSolver](#) > [build](#) (const std::string &target, SharedWavefunctionUnion wfn_union)
Build a solver of a particular property for given molecular cluster.

Protected Attributes

- SharedWavefunctionUnion [wfn_union_](#)
Wavefunction union.
- std::vector< std::string > [methods_oeplib](#)
Names of all OEP-based methods implemented for a solver.
- std::vector< std::string > [methods_benchmark](#)
Names of all benchmark methods implemented for a solver.

12.12.1 Detailed Description

Solver of properties of molecular aggregates. Abstract base.

Uses only a wavefunction union object to initialize.

12.12.2 Constructor & Destructor Documentation

12.12.2.1 OEPDevSolver::OEPDevSolver (SharedWavefunctionUnion wfn_union)

Take wavefunction union and initialize the Solver.

Parameters

<i>wfn_union</i>	- wavefunction union of isolated molecular wavefunctions
------------------	--

12.12.3 Member Function Documentation

12.12.3.1 std::shared_ptr< OEPDevSolver > OEPDevSolver::build (const std::string & target, SharedWavefunctionUnion wfn_union) [static]

Build a solver of a particular property for given molecular cluster.

Parameters

<i>target</i>	- target property
<i>wfn_union</i>	- wavefunction union of isolated molecular wavefunctions

Implemented target properties:

- `ELECTROSTATIC_ENERGY` - Coulombic interaction energy between unperturbed wavefunctions.
- `REPULSION_ENERGY` - Pauli repulsion interaction energy between unperturbed wavefunctions.

See Also

[ElectrostaticEnergySolver](#)

12.12.3.2 `double OEPDevSolver::compute_benchmark (const std::string & method = "DEFAULT")` [pure virtual]

Compute property by using benchmark method.

Each solver object has one `DEFAULT` benchmark method

Parameters

<i>method</i>	- benchmark method
---------------	--------------------

Implemented in [oepdev::RepulsionEnergySolver](#), and [oepdev::ElectrostaticEnergySolver](#).

12.12.3.3 `double OEPDevSolver::compute_oep_based (const std::string & method = "DEFAULT")` [pure virtual]

Compute property by using OEP's.

Each solver object has one `DEFAULT` OEP-based method.

Parameters

<i>method</i>	- flavour of OEP model
---------------	------------------------

Implemented in [oepdev::RepulsionEnergySolver](#), and [oepdev::ElectrostaticEnergySolver](#).

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

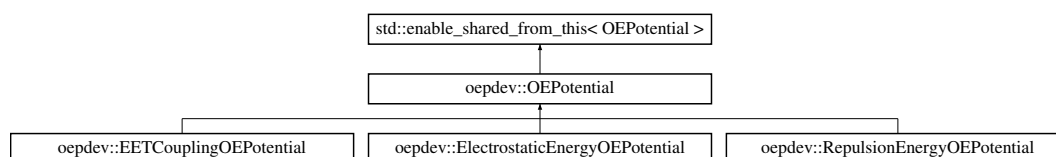
- `oepdev/libutil/solver.h`
- `oepdev/libutil/solver.cc`

12.13 oepdev::OEPotential Class Reference

Generalized One-Electron Potential: Abstract base.

```
#include <oep.h>
```

Inheritance diagram for `oepdev::OEPotential`:



Public Member Functions

- **OEPotential** (SharedWavefunction [wfn](#), Options &options)
ESP-based OEP object.
- **OEPotential** (SharedWavefunction [wfn](#), SharedBasisSet auxiliary, Options &options)
DF-based OEP object.
- virtual **~OEPotential** ()
Destructor.
- virtual void **rotate** (const Matrix &rotmat)
Rotate.
- virtual void **translate** (const Vector &trans)
Translate.
- virtual void **superimpose** (const Matrix &refGeometry, const std::vector< int > &supList, const std::vector< int > &reordList)
Superimpose.
- std::string **name** () const
Retrieve name of this OEP.
- SharedMatrix **matrix** (const std::string &oeptype) const
Retrieve matrix potential.
- SharedWavefunction **wfn** () const
Retrieve wavefunction object.
- void **set_name** (const std::string &name)
- virtual void **print_header** () const =0

- virtual void **compute** (const std::string &oeptype)=0
- virtual void **compute** (void)

- virtual void **write_cube** (const std::string &oeptype, const std::string &fileName)
- virtual void **compute_3D** (const std::string &oeptype, const double &x, const double &y, const double &z, double &v)=0

Static Public Member Functions

- static std::shared_ptr
< **OEPotential** > **build** (const std::string &category, SharedWavefunction [wfn](#), Options &options)
Build ESP-based OEP object.
- static std::shared_ptr
< **OEPotential** > **build** (const std::string &category, SharedWavefunction [wfn](#), SharedBasisSet auxiliary, Options &options)
Build DF-based OEP object.

Public Attributes

- const bool **is_density_fitted**
Is this OEP density-fitted?
- const bool **is_esp_based**
Is this OEP ESP-based?

Protected Attributes

- Options [options_](#)
Psi4 options.
- SharedWavefunction [wfn_](#)
Wavefunction.
- SharedBasisSet [primary_](#)
Promary Basis set.
- SharedBasisSet [auxiliary_](#)
Auxiliary Basis set.
- std::string [name_](#)
Name of this OEP;.
- std::vector< std::string > [oepTypes_](#)
Types of OEP's within the scope of this object.
- std::map< std::string, SharedMatrix > [oepMatrices_](#)
OEP's matrix forms for each OEP type.
- std::shared_ptr< psi::IntegralFactory > [intsFactory_](#)
Integral factory.
- std::shared_ptr< psi::Matrix > [potMat_](#)
Matrix of potential one-electron integrals.
- std::shared_ptr< psi::OneBodyAOInt > [OEInt_](#)
One-electron integral shared pointer.
- std::shared_ptr< [PotentialInt](#) > [potInt_](#)
One-electron potential shared pointer.

12.13.1 Detailed Description

Generalized One-Electron Potential: Abstract base.

Manages OEP's in matrix and 3D forms.

12.13.2 Constructor & Destructor Documentation

12.13.2.1 OEPotential::OEPotential (SharedWavefunction *wfn*, Options & *options*)

ESP-based OEP object.

Parameters

<i>wfn</i>	- wavefunction
<i>options</i>	- Psi4 options

12.13.2.2 OEPotential::OEPotential (SharedWavefunction *wfn*, SharedBasisSet *auxiliary*, Options & *options*)

DF-based OEP object.

Parameters

<i>wfn</i>	- wavefunction
<i>auxiliary</i>	- basis set for density fitting of OEP's
<i>options</i>	- Psi4 options

12.13.3 Member Function Documentation

12.13.3.1 `std::shared_ptr< OEPotential > OEPotential::build (const std::string & category, SharedWavefunction wfn, Options & options) [static]`

Build ESP-based OEP object.

Parameters

<i>type</i>	- OEP category
<i>wfn</i>	- wavefunction
<i>options</i>	- Psi4 options

12.13.3.2 `std::shared_ptr< OEPotential > OEPotential::build (const std::string & category, SharedWavefunction wfn, SharedBasisSet auxiliary, Options & options) [static]`

Build DF-based OEP object.

Parameters

<i>type</i>	- OEP category
<i>wfn</i>	- wavefunction
<i>auxiliary</i>	- basis set for density fitting of OEP's
<i>options</i>	- Psi4 options

12.13.3.3 `void OEPotential::compute_3D (const std::string & oepType, const double & x, const double & y, const double & z, double & v) [pure virtual]`

Compute value of potential in point x, y, z and save at v

Implemented in [oepdev::EETCouplingOEPotential](#), [oepdev::RepulsionEnergyOEPotential](#), and [oepdev::ElectrostaticEnergyOEPotential](#).

12.13.3.4 `void OEPotential::write_cube (const std::string & oepType, const std::string & fileName) [virtual]`

Write potential to a cube file

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

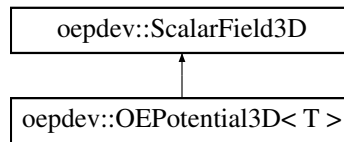
- oepdev/liboep/oep.h
- oepdev/liboep/oep.cc

12.14 oepdev::OEPotential3D< T > Class Template Reference

Class template for OEP scalar fields.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::OEPotential3D< T >:



Public Member Functions

- [OEPotential3D](#) (const int &np, const double &padding, std::shared_ptr< T > oep, const std::string &oepType)
Construct random spherical collection of scalar field of type T.
- [OEPotential3D](#) (const int &nx, const int &ny, const int &nz, const double &px, const double &py, const double &pz, std::shared_ptr< T > oep, const std::string &oepType, psi::Options &options)
Construct ordered 3D collection of scalar field of type T.
- virtual [~OEPotential3D](#) ()
Destructor.
- virtual double [compute_xyz](#) (const double &x, const double &y, const double &z)
Compute a value of scalar field at point (x, y, z)
- virtual void [print](#) () const
Print information of the object to Psi4 output.

Protected Attributes

- std::shared_ptr< T > [oep_](#)
Shared pointer to the instance of class T
- std::string [oepType_](#)
Descriptor of the scalar field type stored in instance of T

Additional Inherited Members

12.14.1 Detailed Description

template<class T>class oepdev::OEPotential3D< T >

Class template for OEP scalar fields.

Used for special type of classes T that contain following public member functions:

```

class T : public std::enable_shared_from_this<T> {
public:
    void compute_3D(const std::string& descriptor,
                  const double& x, const double& y, const double& z,
                  double& v);

    shared_ptr<psi::Wavefunction> wfn() const {return wfn_;}
};
  
```

with the `descriptor` of a certain scalar field type, `x`, `y`, `z` the points in 3D space in which the scalar field has to be computed and stored at `v`. Instances of `T` should store shared pointer to wavefunction object. List of classes `T` that are compatible with this class template and are currently implemented in `oepdev` is given below:

- [oepdev::OEPotential](#) abstract base (do not use derived classes as `T`)

Template parameters:

Template Parameters

<i>T</i>	the compatible class (e.g. <code>oepdev::OEPotential</code>)
----------	---

12.14.2 Constructor & Destructor Documentation

12.14.2.1 `template<class T> oepdev::OEPotential3D< T>::OEPotential3D (const int & np, const double & padding, std::shared_ptr< T> oep, const std::string & oepType)`

Construct random spherical collection of scalar field of type T.

The points are drawn according to uniform distribution in 3D space.

Parameters

<i>np</i>	- number of points to draw
<i>padding</i>	- spherical padding distance (au)
<i>oep</i>	- OEP object of type T
<i>oepType</i>	- type of OEP

12.14.2.2 `template<class T> oepdev::OEPotential3D< T>::OEPotential3D (const int & nx, const int & ny, const int & nz, const double & px, const double & py, const double & pz, std::shared_ptr< T> oep, const std::string & oepType, psi::Options & options)`

Construct ordered 3D collection of scalar field of type T.

The points are generated according to Gaussian cube file format.

Parameters

<i>nx</i>	- number of points along x direction
<i>ny</i>	- number of points along y direction
<i>nz</i>	- number of points along z direction
<i>px</i>	- padding distance along x direction
<i>py</i>	- padding distance along y direction
<i>pz</i>	- padding distance along z direction
<i>oep</i>	- OEP object of type T
<i>oepType</i>	- type of OEP
<i>options</i>	- Psi4 options object

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

- `oepdev/libutil/space3d.h`

12.15 `oepdev::Points3DIterator::Point` Struct Reference

Public Attributes

- double **x**
- double **y**
- double **z**
- int **index**

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

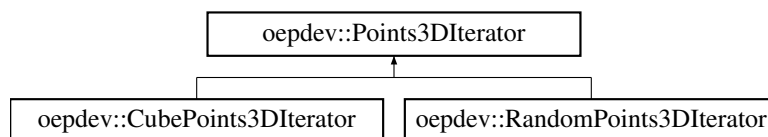
- `oepdev/libutil/space3d.h`

12.16 oepdev::Points3DIterator Class Reference

Iterator over a collection of points in 3D space. Abstract base.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::Points3DIterator:



Classes

- struct [Point](#)

Public Member Functions

- [Points3DIterator](#) (const int &np)
Plain constructor. Initializes the abstract features.
- virtual [~Points3DIterator](#) ()
Destructor.
- virtual bool [is_done](#) ()
Check if iteration is finished.
- virtual void [first](#) ()=0
Initialize first iteration.
- virtual void [next](#) ()=0
Step to next iteration.
- virtual double [x](#) () const
- virtual double [y](#) () const
- virtual double [z](#) () const
- virtual int [index](#) () const

Static Public Member Functions

- static shared_ptr
< [Points3DIterator](#) > [build](#) (const int &nx, const int &ny, const int &nz, const double &dx, const double &dy, const double &dz, const double &ox, const double &oy, const double &oz)
Build G09 Cube collection iterator.
- static shared_ptr
< [Points3DIterator](#) > [build](#) (const int &np, const double &radius, const double &cx, const double &cy, const double &cz)
Build random collection iterator.
- static shared_ptr
< [Points3DIterator](#) > [build](#) (const int &np, const double &pad, psi::SharedMolecule mol)
Build random collection iterator.

Protected Attributes

- const int `np_`
Number of points.
- bool `done_`
Status of the iterator.
- int `index_`
Current index.
- `Point` `current_`

12.16.1 Detailed Description

Iterator over a collection of points in 3D space. Abstract base.

Points3DIterators are constructed either as iterators over:

- a random collections or
- an ordered (g09 cube-like) collections. **Note:** Always create instances by using static factory methods.

12.16.2 Constructor & Destructor Documentation

12.16.2.1 `oepdev::Points3DIterator::Points3DIterator (const int & np)`

Plain constructor. Initializes the abstract features.

Parameters

<code>np</code>	- number of points this iterator is constructed for
-----------------	---

12.16.3 Member Function Documentation

12.16.3.1 `std::shared_ptr< Points3DIterator > oepdev::Points3DIterator::build (const int & nx, const int & ny, const int & nz, const double & dx, const double & dy, const double & dz, const double & ox, const double & oy, const double & oz) [static]`

Build G09 Cube collection iterator.

The points are generated according to Gaussian cube file format.

Parameters

<code>nx</code>	- number of points along x direction
<code>ny</code>	- number of points along y direction
<code>nz</code>	- number of points along z direction
<code>dx</code>	- spacing distance along x direction
<code>dy</code>	- spacing distance along y direction
<code>dz</code>	- spacing distance along y direction
<code>ox</code>	- coordinate x of cube origin
<code>oy</code>	- coordinate y of cube origin
<code>oz</code>	- coordinate z of cube origin

12.16.3.2 `std::shared_ptr< Points3Dlterator > oepdev::Points3Dlterator::build (const int & np, const double & radius, const double & cx, const double & cy, const double & cz) [static]`

Build random collection iterator.

The points are drawn according to uniform distrinution in 3D space.

Parameters

<i>np</i>	- number of points to draw
<i>radius</i>	- sphere radius inside which points are to be drawn
<i>cx</i>	- coordinate x of sphere's centre
<i>cy</i>	- coordinate y of sphere's centre
<i>cz</i>	- coordinate z of sphere's centre

12.16.3.3 `shared_ptr< Points3DIterator > oepdev::Points3DIterator::build (const int & np, const double & pad, psi::SharedMolecule mol) [static]`

Build random collection iterator.

The points are drawn according to uniform distribution in 3D space enclosing a molecule given. All drawn points lie outside the van der Waals volume.

Parameters

<i>np</i>	- number of points to draw
<i>pad</i>	- radius padding of a minimal sphere enclosing the molecule
<i>mol</i>	- Psi4 molecule object

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

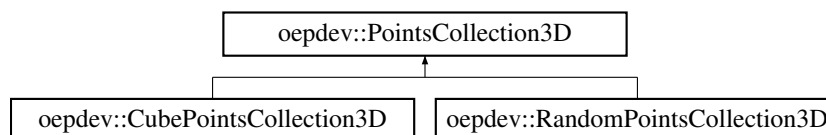
- oepdev/libutil/space3d.h
- oepdev/libutil/space3d.cc

12.17 oepdev::PointsCollection3D Class Reference

Collection of points in 3D space. Abstract base.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::PointsCollection3D:



Public Types

- enum [Collection](#) { **Random**, **Cube** }
- Public descriptor of collection type.*

Public Member Functions

- [PointsCollection3D](#) ([Collection](#) collectionType, int &np)
Initialize abstract features.
- **PointsCollection3D** ([Collection](#) collectionType, const int &np)
- virtual `~PointsCollection3D ()`
Destructor.
- virtual int [npoints](#) () const
Get the number of points.

- virtual shared_ptr
< Points3DIterator > points_iterator () const
Get the iterator over this collection of points.
- virtual Collection get_type () const
Get the collection type.
- virtual void print () const =0
Print the information to Psi4 output file.

Static Public Member Functions

- static shared_ptr
< PointsCollection3D > build (const int &npoints, const double &radius, const double &cx=0.0, const double &cy=0.0, const double &cz=0.0)
Build random collection of points.
- static shared_ptr
< PointsCollection3D > build (const int &npoints, const double &padding, psi::SharedMolecule mol)
Build random collection of points.
- static shared_ptr
< PointsCollection3D > build (const int &nx, const int &ny, const int &nz, const double &px, const double &py, const double &pz, psi::SharedBasisSet bs, psi::Options &options)
Build G09 Cube collection of points.

Protected Attributes

- const int np_
Number of points.
- Collection collectionType_
Collection type.
- shared_ptr< Points3DIterator > pointsIterator_
iterator over points collection

12.17.1 Detailed Description

Collection of points in 3D space. Abstract base.

Create random or ordered (g09 cube-like) collections of points in 3D space.

Note: Always create instances by using static factory methods.

12.17.2 Constructor & Destructor Documentation

12.17.2.1 oepdev::PointsCollection3D::PointsCollection3D (Collection collectionType, int & np)

Initialize abstract features.

Parameters

<i>np</i>	- number of points to be created
-----------	----------------------------------

12.17.3 Member Function Documentation

12.17.3.1 `std::shared_ptr< PointsCollection3D > oepdev::PointsCollection3D::build (const int & npoints, const double & radius, const double & cx = 0.0, const double & cy = 0.0, const double & cz = 0.0) [static]`

Build random collection of points.

Points uniformly span a sphere.

Parameters

<i>npoints</i>	- number of points to draw
<i>radius</i>	- sphere radius inside which points are to be drawn
<i>cx</i>	- coordinate x of sphere's centre
<i>cy</i>	- coordinate y of sphere's centre
<i>cz</i>	- coordinate z of sphere's centre

12.17.3.2 `std::shared_ptr< PointsCollection3D > oepdev::PointsCollection3D::build (const int & npoints, const double & padding, psi::SharedMolecule mol) [static]`

Build random collection of points.

Points uniformly span space inside a sphere enclosing a molecule. excluding the van der Waals volume.

Parameters

<i>np</i>	- number of points to draw
<i>padding</i>	- radius padding of a minimal sphere enclosing the molecule
<i>mol</i>	- Psi4 molecule object

12.17.3.3 `std::shared_ptr< PointsCollection3D > oepdev::PointsCollection3D::build (const int & nx, const int & ny, const int & nz, const double & px, const double & py, const double & pz, psi::SharedBasisSet bs, psi::Options & options) [static]`

Build G09 Cube collection of points.

The points span a parallelepiped according to Gaussian cube file format.

Parameters

<i>nx</i>	- number of points along x direction
<i>ny</i>	- number of points along y direction
<i>nz</i>	- number of points along z direction
<i>px</i>	- padding distance along x direction
<i>py</i>	- padding distance along y direction
<i>pz</i>	- padding distance along z direction
<i>bs</i>	- Psi4 basis set object
<i>options</i>	- Psi4 options object

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

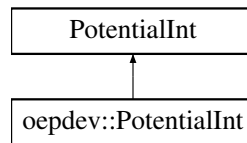
- oepdev/libutil/space3d.h
- oepdev/libutil/space3d.cc

12.18 oepdev::PotentialInt Class Reference

Computes potential integrals.


```
#include <potential.h>
```

Inheritance diagram for oepdev::PotentialInt:



Public Member Functions

- [PotentialInt](#) (std::vector< psi::SphericalTransform > &st, std::shared_ptr< psi::BasisSet > bs1, std::shared_ptr< psi::BasisSet > bs2, int deriv=0)
Constructor. Initialize identically like in psi::Potentillnt.
- [PotentialInt](#) (std::vector< psi::SphericalTransform > &st, std::shared_ptr< psi::BasisSet > bs1, std::shared_ptr< psi::BasisSet > bs2, std::shared_ptr< psi::Matrix > Qxyz, int deriv=0)
Constructor. Takes an arbitrary collection of charges.
- [PotentialInt](#) (std::vector< psi::SphericalTransform > &, std::shared_ptr< psi::BasisSet >, std::shared_ptr< psi::BasisSet >, const double &x, const double &y, const double &z, const double &q=1.0, int deriv=0)
Constructor. Computes potential for one point x, y, z for a test particle of charge q.
- void [set_charge_field](#) (const double &x, const double &y, const double &z, const double &q=1.0)
Mutator. Set the charge field to be a x, y, z point of charge q.

12.18.1 Detailed Description

Computes potential integrals.

12.18.2 Constructor & Destructor Documentation

12.18.2.1 oepdev::PotentialInt::PotentialInt (std::vector< psi::SphericalTransform > &st, std::shared_ptr< psi::BasisSet > bs1, std::shared_ptr< psi::BasisSet > bs2, int deriv = 0)

Constructor. Initialize identically like in psi::Potentillnt.

Parameters

<i>st</i>	- Spherical transform object
<i>bs1</i>	- basis set for first space
<i>bs2</i>	- basis set for second space
<i>deriv</i>	- derivative level

12.18.2.2 oepdev::PotentialInt::PotentialInt (std::vector< psi::SphericalTransform > &st, std::shared_ptr< psi::BasisSet > bs1, std::shared_ptr< psi::BasisSet > bs2, std::shared_ptr< psi::Matrix > Qxyz, int deriv = 0)

Constructor. Takes an arbitrary collection of charges.

Parameters

<i>st</i>	- Spherical transform object
-----------	------------------------------

<i>bs1</i>	- basis set for first space
<i>bs2</i>	- basis set for second space
<i>Qxyz</i>	- matrix with charges and their positions
<i>deriv</i>	- derivative level

12.18.2.3 `oepdev::PotentialInt::PotentialInt (std::vector< psi::SphericalTransform > & st, std::shared_ptr< psi::BasisSet > bs1, std::shared_ptr< psi::BasisSet > bs2, const double & x, const double & y, const double & z, const double & q = 1.0, int deriv = 0)`

Constructor. Computes potential for one point *x*, *y*, *z* for a test particle of charge *q*.

Parameters

<i>st</i>	- Spherical transform object
<i>bs1</i>	- basis set for first space
<i>bs2</i>	- basis set for second space
<i>x</i>	- x coordinate of <i>q</i>
<i>y</i>	- y coordinate of <i>q</i>
<i>z</i>	- z coordinate of <i>q</i>
<i>q</i>	- value of the probe charge
<i>deriv</i>	- derivative level

12.18.3 Member Function Documentation

12.18.3.1 `void oepdev::PotentialInt::set_charge_field (const double & x, const double & y, const double & z, const double & q = 1.0)`

Mutator. Set the charge field to be a *x*, *y*, *z* point of charge *q*.

Parameters

<i>x</i>	- x coordinate of <i>q</i>
<i>y</i>	- y coordinate of <i>q</i>
<i>z</i>	- z coordinate of <i>q</i>
<i>q</i>	- value of the probe charge

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

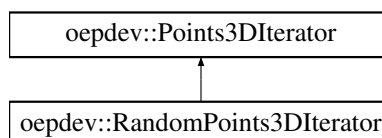
- oepdev/libpsi/potential.h
- oepdev/libpsi/potential.cc

12.19 oepdev::RandomPoints3DIterator Class Reference

Iterator over a collection of points in 3D space. Random collection.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::RandomPoints3DIterator:



Public Member Functions

- **RandomPoints3DIterator** (const int &np, const double &radius, const double &cx, const double &cy, const double &cz)
- **RandomPoints3DIterator** (const int &np, const double &pad, psi::SharedMolecule mol)
- virtual void [first](#) ()
Initialize first iteration.
- virtual void [next](#) ()
Step to next iteration.

Protected Member Functions

- virtual double **random_double** ()
- virtual void **draw_random_point** ()
- virtual bool **is_in_vdWsphere** (double x, double y, double z) const

Protected Attributes

- double **cx_**
- double **cy_**
- double **cz_**
- double **radius_**
- double **r_**
- double **phi_**
- double **theta_**
- double **x_**
- double **y_**
- double **z_**
- psi::SharedMatrix **excludeSpheres_**
- std::map< std::string, double > **vdwRadius_**
- std::default_random_engine **randomNumberGenerator_**
- std::uniform_real_distribution
< double > **randomDistribution_**

Additional Inherited Members

12.19.1 Detailed Description

Iterator over a collection of points in 3D space. Random collection.

Note: Always create instances by using static factory method from [Points3DIterator](#). Do not use constructors of this class.

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

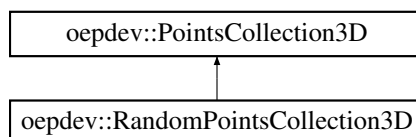
- oepdev/libutil/space3d.h
- oepdev/libutil/space3d.cc

12.20 oepdev::RandomPointsCollection3D Class Reference

Collection of random points in 3D space.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::RandomPointsCollection3D:



Public Member Functions

- **RandomPointsCollection3D** ([Collection](#) collectionType, const int &npoints, const double &radius, const double &cx, const double &cy, const double &cz)
- **RandomPointsCollection3D** ([Collection](#) collectionType, const int &npoints, const double &padding, psi::SharedMolecule mol)
- virtual void [print](#) () const

Print the information to Psi4 output file.

Additional Inherited Members

12.20.1 Detailed Description

Collection of random points in 3D space.

Note: Do not use constructors of this class explicitly. Instead, use static factory methods of the superclass to create instances.

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

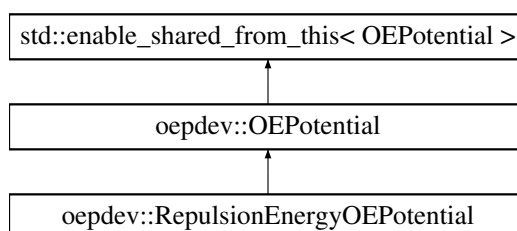
- oepdev/libutil/space3d.h
- oepdev/libutil/space3d.cc

12.21 oepdev::RepulsionEnergyOEPotential Class Reference

Generalized One-Electron Potential for Pauli repulsion energy calculations.

```
#include <oep.h>
```

Inheritance diagram for oepdev::RepulsionEnergyOEPotential:



Public Member Functions

- **RepulsionEnergyOEPotential** (SharedWavefunction [wfn](#), SharedBasisSet auxiliary, Options &options)
- **RepulsionEnergyOEPotential** (SharedWavefunction [wfn](#), Options &options)
- virtual void **compute** (const std::string &oepType) override
- virtual void **compute_3D** (const std::string &oepType, const double &x, const double &y, const double &z, double &v) override
- virtual void **print_header** () const override

Additional Inherited Members

12.21.1 Detailed Description

Generalized One-Electron Potential for Pauli repulsion energy calculations.

Contains the following OEP types:

12.21.2 Member Function Documentation

12.21.2.1 void RepulsionEnergyOEPotential::compute_3D (const std::string & oepType, const double & x, const double & y, const double & z, double & v) [override],[virtual]

Compute value of potential in point x, y, z and save at v

Implements [oepdev::OEPotential](#).

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

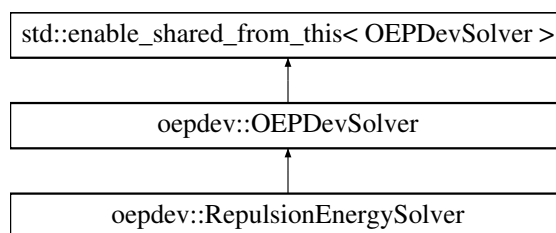
- oepdev/liboep/oep.h
- oepdev/liboep/oep.cc

12.22 oepdev::RepulsionEnergySolver Class Reference

Compute the Pauli-Repulsion interaction energy between unperturbed wavefunctions.

```
#include <solver.h>
```

Inheritance diagram for oepdev::RepulsionEnergySolver:



Public Member Functions

- **RepulsionEnergySolver** (SharedWavefunctionUnion wfn_union)
- virtual double **compute_oep_based** (const std::string &method="DEFAULT")
Compute property by using OEP's.
- virtual double **compute_benchmark** (const std::string &method="DEFAULT")
Compute property by using benchmark method.

Additional Inherited Members

12.22.1 Detailed Description

Compute the Pauli-Repulsion interaction energy between unperturbed wavefunctions.

The implemented methods are shown below

Keyword	Method Description
Benchmark Methods	
HAYES_STONE	*Default*. Pauli Repulsion energy at HF level from Hayes and Stone (1984).
DENSITY_BASED	Pauli Repulsion energy at HF level from Mandado and Hermida-Ramon (2012).
MURRELL_ETAL	Approximate Pauli Repulsion energy at HF level from Murrell et al.
EFP2	Approximate Pauli Repulsion energy at HF level from EFP2 model.
OEP-Based Methods	
MURRELL_ETAL_MIX	*Default*. OEP-Murrell et al's: S1 term via DF-OEP, S2 term via ESP-OEP.
MURRELL_ETAL_ESP	OEP-Murrell et al's: S1 and S2 via ESP-OEP

Table 12.2: Methods available in the Solver

Below the detailed description of the above methods is given. In the formulae, the Coulomb notation for electron repulsion integrals (ERI's) in MO basis is adopted; i.e.,

$$(ac|bd) = \iint d\mathbf{r}_1 d\mathbf{r}_2 \phi_a(\mathbf{r}_1) \phi_c(\mathbf{r}_1) \frac{1}{r_{12}} \phi_b(\mathbf{r}_2) \phi_d(\mathbf{r}_2)$$

It is also assumed that the orbitals are real.

Benchmark Methods

Pauli Repulsion energy at HF level by Hayes and Stone (1984).

For a closed-shell system, equation of Hayes and Stone (1984) becomes

$$E^{\text{Rep}} = 2 \sum_{kl} (V_{kl}^A + V_{kl}^B + T_{kl}) [[\mathbf{S}^{-1}]_{lk} - \delta_{lk}] + \sum_{klmn} (kl|mn) \{ 2[\mathbf{S}^{-1}]_{kl}[\mathbf{S}^{-1}]_{mn} - [\mathbf{S}^{-1}]_{kn}[\mathbf{S}^{-1}]_{lm} - 2\delta_{kl}\delta_{mn} + \delta_{kn}\delta_{lm} \}$$

where \mathbf{S} is the overlap matrix between the doubly-occupied orbitals. The exact, pure exchange energy is for a closed shell case given as

$$E^{\text{Ex,pure}} = -2 \sum_{a \in A} \sum_{b \in B} (ab|ba)$$

Similarity transformation of molecular orbitals does not affect the resulting energies. The overall exchange-repulsion interaction energy is then (always net repulsive)

$$E^{\text{Ex-Rep}} = E^{\text{Ex,pure}} + E^{\text{Rep}}$$

Repulsion energy of Mandado and Hermida-Ramon (2011)

At the Hartree-Fock level, the exchange-repulsion energy from the density-based scheme of Mandado and Hermida-Ramon (2011) is fully equivalent to the method by Hayes and Stone (1984). However, density-based method

enables to compute exchange-repulsion energy at any level of theory. It is derived based on the Pauli deformation density matrix,

$$\Delta \mathbf{D}^{\text{Pauli}} \equiv \mathbf{D}^{oo} - \mathbf{D}$$

where \mathbf{D}^{oo} and \mathbf{D} are the density matrix formed from mutually orthogonal sets of molecular orbitals within the entire aggregate (formed by symmetric orthogonalization of MO's) and the density matrix of the unperturbed system (that can be understood as a Hadamard sum $\mathbf{D} \equiv \mathbf{D}^A \oplus \mathbf{D}^B$).

At HF level, the Pauli deformation density matrix is given by

$$\Delta \mathbf{D}^{\text{Pauli}} = \mathbf{C} [\mathbf{S}^{-1} - \mathbf{1}] \mathbf{C}^\dagger$$

whereas the density matrix constructed from mutually orthogonal orbitals is

$$\mathbf{D}^{oo} = \mathbf{C} \mathbf{S}^{-1} \mathbf{C}^\dagger$$

In the above equations, \mathbf{S} is the overlap matrix between doubly occupied molecular orbitals of the entire aggregate.

Here, the expressions for the exchange-repulsion energy at any level of theory are shown for the case of open-shell system. The net repulsive energy is given as

$$E^{\text{Ex-Rep}} = E^{\text{Rep},1} + E^{\text{Rep},2} + E^{\text{Ex}}$$

where the one- and two-electron part of the repulsion energy is

$$E^{\text{Rep},1} = E^{\text{Rep},\text{Kin}} + E^{\text{Rep},\text{Nuc}}$$

$$E^{\text{Rep},2} = E^{\text{Rep},\text{el}-\Delta} + E^{\text{Rep},\Delta-\Delta}$$

The kinetic and nuclear contributions are

$$E^{\text{Rep},\text{Kin}} = 2 \sum_{\alpha\beta \in A,B} \Delta D_{\alpha\beta}^{\text{Pauli}} T_{\alpha\beta}$$

$$E^{\text{Rep},\text{Nuc}} = 2 \sum_{\alpha\beta \in A,B} \Delta D_{\alpha\beta}^{\text{Pauli}} \sum_{z \in A,B} V_{\alpha\beta}^{(z)}$$

whereas the electron-deformation and deformation-deformation interaction contributions are

$$E^{\text{Rep},\text{el}-\Delta} = 4 \sum_{\alpha\beta\gamma\delta \in A,B} \Delta D_{\alpha\beta}^{\text{Pauli}} D_{\gamma\delta} (\alpha\beta|\gamma\delta)$$

$$E^{\text{Rep},\Delta-\Delta} = 2 \sum_{\alpha\beta\gamma\delta \in A,B} \Delta D_{\alpha\beta}^{\text{Pauli}} \Delta D_{\gamma\delta}^{\text{Pauli}} (\alpha\beta|\gamma\delta)$$

The associated exchange energy is given by

$$E^{\text{Ex}} = - \sum_{\alpha\beta\gamma\delta \in A,B} \left[D_{\alpha\delta}^{oo} D_{\beta\gamma}^{oo} - D_{\alpha\delta}^A D_{\beta\gamma}^A - D_{\alpha\delta}^B D_{\beta\gamma}^B \right] (\alpha\beta|\gamma\delta)$$

It is important to emphasise that, although, at HF level, the particular 'repulsive' and 'exchange' energies computed by using either Hayes and Stone or Mandado and Hermida-Ramon methods are not equal to each other, they sum up to exactly the same exchange-repulsion energy, $E^{\text{Ex-Rep}}$. Therefore, these methods at HF level are fully equivalent but the nature of partitioning of repulsive and exchange parts is different. It is also noted that the orbital localization does *not* affect the resulting energies, as opposed to the next approximate methods described below.

Approximate Pauli Repulsion energy at HF level from Murrell et al.

By expanding the overlap matrix in a Taylor series one can show that the Pauli repulsion energy is approximately given as

$$E^{\text{Rep}} = E^{\text{Rep}}(\mathcal{O}(S)) + E^{\text{Rep}}(\mathcal{O}(S^2))$$

where the first-order term is

$$E^{\text{Rep}}(\mathcal{O}(S)) = -2 \sum_{a \in A} \sum_{b \in B} S_{ab} \left\{ V_{ab}^A + \sum_{c \in A} [2(ab|cc) - (ac|bc)] + V_{ab}^B + \sum_{d \in B} [2(ab|dd) - (ad|bd)] \right\}$$

whereas the second-order term is

$$E^{\text{Rep}}(\mathcal{O}(S^2)) = 2 \sum_{a \in A} \sum_{b \in B} S_{ab} \left\{ \sum_{c \in A} S_{bc} \left[V^B + 2 \sum_{d \in B} (ac|dd) \right] + \sum_{d \in B} S_{ad} \left[V^A + 2 \sum_{x \in A} (bd|cc) \right] - \sum_{c \in A} \sum_{d \in B} S_{cd} (ac|bd) \right\}$$

By using OEP technique, the above theory can be exactly re-cast *without* any further approximations.

OEP-Based Methods

The Murrell et al's theory of Pauli repulsion is here re-cast by introducing OEP's.

S1 term via DF-OEP, S2 term via ESP-OEP.

S1 and S2 terms via ESP-OEP.

Notes:

- This solver also computes and prints the exchange energy at HF level (formula is given above) for reference purposes.
- In order to construct this solver, **always** use the `OEPDevSolver::build` static factory method.

12.22.2 Member Function Documentation

12.22.2.1 `double RepulsionEnergySolver::compute_benchmark (const std::string & method = "DEFAULT")`
[virtual]

Compute property by using benchmark method.

Each solver object has one `DEFAULT` benchmark method

Parameters

<i>method</i>	- benchmark method
---------------	--------------------

Implements `oepdev::OEPDevSolver`.

12.22.2.2 `double RepulsionEnergySolver::compute_oep_based (const std::string & method = "DEFAULT")`
[virtual]

Compute property by using OEP's.

Each solver object has one `DEFAULT` OEP-based method.

Parameters

<i>method</i>	- flavour of OEP model
---------------	------------------------

Implements `oepdev::OEPDevSolver`.

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

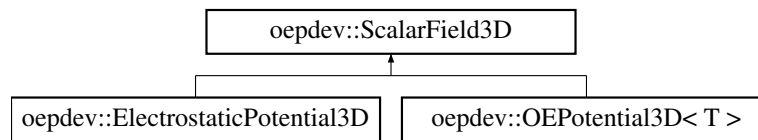
- `oepdev/libutil/solver.h`
- `oepdev/libutil/solver.cc`

12.23 `oepdev::ScalarField3D` Class Reference

Scalar field in 3D space. Abstract base.

```
#include <space3d.h>
```

Inheritance diagram for `oepdev::ScalarField3D`:



Public Member Functions

- [ScalarField3D](#) (const int &np, const double &pad, psi::SharedWavefunction [wfn](#), psi::Options &options)
Construct potential on random grid by providing wavefunction.
- [ScalarField3D](#) (const int &nx, const int &ny, const int &nz, const double &px, const double &py, const double &pz, std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &options)
Construct potential on cube grid by providing wavefunction.
- virtual [~ScalarField3D](#) ()
Destructor.
- virtual int [npoints](#) () const
Get the number of points at which the scalar field is defined.
- virtual std::shared_ptr< [PointsCollection3D](#) > [points_collection](#) () const
Get the collection of points.
- virtual std::shared_ptr< psi::Matrix > [data](#) () const
Get the data matrix in a form { [x, y, z, f(x, y, z)] }.
- virtual std::shared_ptr< psi::Wavefunction > [wfn](#) () const
Get the wavefunction.
- virtual bool [is_computed](#) () const
Get the information if data is already computed or not.
- virtual void [compute](#) ()
Compute the scalar field in each point from the point collection.
- virtual double [compute_xyz](#) (const double &x, const double &y, const double &z)=0
Compute a value of scalar field at point (x, y, z)
- virtual void [write_cube_file](#) (const std::string &name)
Write the cube file (only for Cube collections, otherwise does nothing)
- virtual void [print](#) () const =0
Print information of the object to Psi4 output.

Static Public Member Functions

- static shared_ptr< [ScalarField3D](#) > [build](#) (const std::string &type, const int &np, const double &pad, psi::SharedWavefunction [wfn](#), psi::Options &options)
Build scalar field of random points.
- static shared_ptr< [ScalarField3D](#) > [build](#) (const std::string &type, const int &nx, const int &ny, const int &nz, const double &px, const double &py, const double &pz, psi::SharedWavefunction [wfn](#), psi::Options &options)
Build scalar field of points on a g09-cube grid.

Protected Attributes

- `std::shared_ptr`
`< PointsCollection3D > pointsCollection_`
Collection of points at which the scalar field is to be computed.
- `std::shared_ptr< psi::Matrix >` `data_`
The data matrix in a form $\{ [x, y, z, f(x, y, z)] \}$.
- `std::shared_ptr`
`< psi::Wavefunction > wfn_`
Wavefunction.
- `psi::Matrix` `geom_`
Geometry of a molecule.
- `std::shared_ptr`
`< psi::IntegralFactory > fact_`
Integral factory.
- `std::shared_ptr< psi::Matrix >` `pot_`
Matrix of potential one-electron integrals.
- `std::shared_ptr`
`< psi::OneBodyAOInt > oneInt_`
One-electron integral shared pointer.
- `std::shared_ptr< PotentialInt >` `potInt_`
One-electron potential shared pointer.
- `std::shared_ptr< psi::BasisSet >` `primary_`
Basis set.
- `int` `nbf_`
Number of basis functions.
- `bool` `isComputed_`
Has data already computed?

12.23.1 Detailed Description

Scalar field in 3D space. Abstract base.

Create scalar field defined at points distributed randomly or as an ordered g09 cube-like collection. Currently implemented scalar fields are:

- Electrostatic potential - computes electrostatic potential (requires wavefunction)
- Template of generic classes - compute custom scalar fields (requires generic object that is able to compute the field in 3D space)

Note: Always create instances by using static factory methods `build`. The following types of scalar field are currently implemented:

- ELECTROSTATIC POTENTIAL

12.23.2 Member Function Documentation

12.23.2.1 `std::shared_ptr< ScalarField3D > oepdev::ScalarField3D::build (const std::string & type, const int & np, const double & pad, psi::SharedWavefunction wfn, psi::Options & options) [static]`

Build scalar field of random points.

Parameters

<i>type</i>	- type of scalar field
<i>np</i>	- number of points
<i>pad</i>	- radius padding of a minimal sphere enclosing the molecule
<i>wfn</i>	- Psi4 Wavefunction containing the molecule
<i>options</i>	- Psi4 options

12.23.2.2 `std::shared_ptr< ScalarField3D > oepdev::ScalarField3D::build (const std::string & type, const int & nx, const int & ny, const int & nz, const double & px, const double & py, const double & pz, psi::SharedWavefunction wfn, psi::Options & options) [static]`

Build scalar field of points on a g09-cube grid.

Parameters

<i>type</i>	- type of scalar field
<i>nx</i>	- number of points along x direction
<i>ny</i>	- number of points along y direction
<i>nz</i>	- number of points along z direction
<i>px</i>	- padding distance along x direction
<i>py</i>	- padding distance along y direction
<i>pz</i>	- padding distance along z direction
<i>wfn</i>	- Psi4 Wavefunction containing the molecule
<i>options</i>	- Psi4 options

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

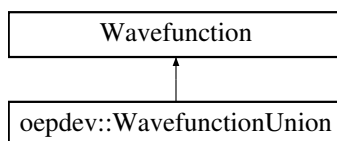
- oepdev/libutil/space3d.h
- oepdev/libutil/space3d.cc

12.24 oepdev::WavefunctionUnion Class Reference

Union of two Wavefunction objects.

```
#include <wavefunction_union.h>
```

Inheritance diagram for oepdev::WavefunctionUnion:



Public Member Functions

- [WavefunctionUnion](#) (SharedWavefunction ref_wfn, Options &options)
Constructor.
- virtual [~WavefunctionUnion](#) ()
Destructor.
- virtual double [compute_energy](#) ()
Compute Energy (now blank)
- virtual double [nuclear_repulsion_interaction_energy](#) ()
Compute Nuclear Repulsion Energy between unions.

- void [localize_orbitals](#) ()
Localize Molecular Orbitals.
- void [transform_integrals](#) ()
Transform Integrals (2- and 4-index transformations)
- int [l_nmo](#) (int n) const
*Get number of molecular orbitals of the *n*th fragment.*
- int [l_nso](#) (int n) const
*Get number of symmetry orbitals of the *n*th fragment.*
- int [l_ndocc](#) (int n) const
*Get number of doubly occupied orbitals of the *n*th fragment.*
- int [l_nvir](#) (int n) const
*Get number of virtual orbitals of the *n*th fragment.*
- int [l_nalpha](#) (int n) const
*Get the number of the alpha electrons of the *n*th fragment.*
- int [l_nbeta](#) (int n) const
*Get the number of the beta electrons of the *n*th fragment.*
- int [l_nbf](#) (int n) const
*Get number of basis functions of the *n*th fragment.*
- int [l_noffs_ao](#) (int n) const
*Get the basis set offset of the *n*th fragment.*
- double [l_energy](#) (int n) const
*Get the reference energy of the *n*th fragment.*
- SharedMolecule [l_molecule](#) (int n) const
*Get the molecule object of the *n*th fragment.*
- SharedBasisSet [l_primary](#) (int n) const
*Get the primary basis set object of the *n*th fragment.*
- SharedBasisSet [l_auxiliary](#) (int n) const
*Get the auxiliary basis set object of the *n*th fragment.*
- SharedWavefunction [l_wfn](#) (int n) const
*Get the wavefunction object of the *n*th fragment.*
- SharedMOSpace [l_mospace](#) (int n, const std::string &label) const
*Get the MO space named `label` (either `OCC` or `VIR`) of the *n*th fragment.*
- SharedLocalizer [l_localizer](#) (int n) const
*Get the orbital localizer object of the *n*th fragment.*
- SharedIntegralTransform [integrals](#) (void) const
Get the integral transform object of the entire union.
- bool [has_localized_orbitals](#) (void) const
If union got its molecular orbital localized or not.
- SharedBasisSet [primary](#) (void) const
Get the primary basis set for the entire union.
- SharedMOSpace [mospace](#) (const std::string &label) const
Get the MO space named `label` (either `OCC` or `VIR`)
- SharedMatrix [Ca_subset](#) (const std::string &basis="SO", const std::string &subset="ALL")
- SharedMatrix [Cb_subset](#) (const std::string &basis="SO", const std::string &subset="ALL")
- SharedMatrix [C_subset_helper](#) (SharedMatrix C, const Dimension &noccpi, SharedVector epsilon, const std::string &basis, const std::string &subset)
Helpers for `Ca_` and `Cb_` matrix transformers.
- SharedVector [epsilon_subset_helper](#) (SharedVector epsilon, const Dimension &noccpi, const std::string &basis, const std::string &subset)
Helper for `epsilon` transformer.
- void [print_header](#) (void)
Print information about this wavefunction union.
- void [print_mo_integrals](#) (void)
Print the MO integrals.

Protected Attributes

- int [nIsolatedMolecules_](#)
Number of isolated molecules.
- SharedWavefunction [dimer_wavefunction_](#)
The wavefunction for a dimer (electrons relaxed in the field of monomers)
- SharedIntegralTransform [integrals_](#)
Integral transform object (2- and 4-index transformations)
- bool [hasLocalizedOrbitals_](#)
whether orbitals of the union were localized (or not)
- std::map< const std::string, SharedMOSpace > [mospacesUnion_](#)
Dictionary of MO spaces for the entire union (OCC and VIR)
- std::vector< SharedMolecule > [l_molecule_](#)
List of molecules.
- std::vector< SharedBasisSet > [l_primary_](#)
List of primary basis functions per molecule.
- std::vector< SharedBasisSet > [l_auxiliary_](#)
List of auxiliary basis functions per molecule.
- std::vector< SharedWavefunction > [l_wfn_](#)
List of original isolated wavefunctions (electrons unrelaxed)
- std::vector< std::string > [l_name_](#)
List of names of isolated wavefunctions.
- std::vector< int > [l_nbf_](#)
List of basis function numbers per molecule.
- std::vector< int > [l_nmo_](#)
List of numbers of molecular orbitals (MO's) per molecule.
- std::vector< int > [l_nso_](#)
List of numbers of SO's per molecule.
- std::vector< int > [l_ndocc_](#)
List of numbers of doubly occupied orbitals per molecule.
- std::vector< int > [l_nvir_](#)
List of numbers of virtual orbitals per molecule.
- std::vector< int > [l_noffs_ao_](#)
List of basis set offsets per molecule.
- std::vector< double > [l_energy_](#)
List of energies of isolated wavefunctions.
- std::vector< double > [l_efzc_](#)
List of frozen-core energies per isolated wavefunction.
- std::vector< bool > [l_density_fitted_](#)
List of information per wfn whether it was obtained using DF or not.
- std::vector< int > [l_nalpha_](#)
List of numbers of alpha electrons per isolated wavefunction.
- std::vector< int > [l_nbeta_](#)
List of numbers of beta electrons per isolated wavefunction.
- std::vector< int > [l_nfrzc_](#)
List of numbers of frozen-core orbitals per isolated molecule.
- std::vector< SharedLocalizer > [l_localizer_](#)
List of orbital localizers.
- std::vector< std::map< const std::string, SharedMOSpace > > [l_mospace_](#)
List of dictionaries of MO spaces.

12.24.1 Detailed Description

Union of two Wavefunction objects.

The [WavefunctionUnion](#) is the union of two unperturbed Wavefunctions.

Notes:

1. Works only for C1 symmetry! Therefore `this->nirrep() = 1`.
2. Does not set `reference_wavefunction_`
3. Sets `oeprop_` for the union of uncoupled molecules
4. Performs Hadamard sums on `H_`, `Fa_`, `Da_`, `Ca_` and `S_` based on uncoupled wavefunctions.
5. Since it is based on shallow copy of the original Wavefunction, it **changes** contents of this wavefunction. Reallocate and copy if you want to keep the original wavefunction.

Warnings:

1. Gradients, Hessians and frequencies are not touched, hence they are **wrong**!
2. Lagrangian (if present) is not touched, hence its **wrong**!
3. Ca/Cb and epsilon subsets were reimplemented from `psi::Wavefunction` to remove sorting of orbitals. However, the corresponding member functions are not virtual in `psi::Wavefunction`. This could bring problems when upcasting.

The following variables are *shallow* copies of variables inside the Wavefunction object, that is created for the *whole* molecule cluster:

- `basissets_` (DF/RI/F12/etc basis sets)_
- `basisset_` (ORBITAL basis set)
- `sobasisset_` (Primary basis set for SO integrals)
- `AO2SO_` (AO2SO conversion matrix (AO in rows, SO in cols)
- `molecule_` (Molecule that this wavefunction is run on)
- `options_` (Options object)
- `psio_` (PSI file access variables)
- `integral_` (Integral factory)
- `factory_` (Matrix factory for creating standard sized matrices)
- `memory_` (How much memory you have access to)
- `nalpha_, nbeta_` (Total alpha and beta electrons)
- `nfrzc_` (Total frozen core orbitals)
- `doccpi_` (Number of doubly occupied per irrep)
- `soccpi_` (Number of singly occupied per irrep)
- `frzcpi_` (Number of frozen core per irrep)
- `frzvp_` (Number of frozen virtuals per irrep)
- `nalphapi_` (Number of alpha electrons per irrep)
- `nbetapi_` (Number of beta electrons per irrep)

- `nsopi_` (Number of so per irrep)
- `nmopi_` (Number of mo per irrep)
- `nso_` (Total number of SOs)
- `nmo_` (Total number of MOs)
- `nirrep_` (Number of irreps; must be equal to 1 due to symmetry reasons)
- `same_a_b_dens_` and `same_a_b_orbs_` The rest is altered so that the Wavefunction parameters reflect a cluster of non-interacting (uncoupled, isolated, unrelaxed) molecular electron densities.

12.24.2 Constructor & Destructor Documentation

12.24.2.1 oepdev::WavefunctionUnion::WavefunctionUnion (SharedWavefunction *ref_wfn*, Options & *options*)

Constructor.

Provide wavefunction with molecule containing at least 2 fragments.

Parameters

<i>ref_wfn</i>	- reference wavefunction
<i>options</i>	- Psi4 options

12.24.3 Member Function Documentation

12.24.3.1 SharedMatrix oepdev::WavefunctionUnion::Ca_subset (const std::string & *basis* = "SO", const std::string & *subset* = "ALL")

Return a subset of the Ca matrix in a desired basis

Parameters

<i>basis</i>	the symmetry basis to use AO, SO
<i>subset</i>	the subset of orbitals to return ALL, ACTIVE, FROZEN, OCC, VIR, FROZEN_OCC, ACTIVE_OCC, ACTIVE_VIR, FROZEN_VIR

Returns

the matrix in Pitzer order in the desired basis

12.24.3.2 SharedMatrix oepdev::WavefunctionUnion::Cb_subset (const std::string & *basis* = "SO", const std::string & *subset* = "ALL")

Return a subset of the Cb matrix in a desired basis

Parameters

<i>basis</i>	the symmetry basis to use AO, SO
<i>subset</i>	the subset of orbitals to return ALL, ACTIVE, FROZEN, OCC, VIR, FROZEN_OCC, ACTIVE_OCC, ACTIVE_VIR, FROZEN_VIR

Returns

the matrix in Pitzer order in the desired basis

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

- `oepdev/libutil/wavefunction_union.h`
- `oepdev/libutil/wavefunction_union.cc`

Index

- AllAOIntegralsIterator
 - oepdev::AllAOIntegralsIterator, [30](#)
- AllAOShellCombinationsIterator
 - oepdev::AllAOShellCombinationsIterator, [31](#)
- build
 - oepdev::OEPDevSolver, [44](#)
 - oepdev::OEPotential, [48](#)
 - oepdev::Points3DIterator, [52](#), [54](#)
 - oepdev::PointsCollection3D, [56](#)
 - oepdev::ScalarField3D, [66](#), [67](#)
- CPHF
 - oepdev::CPHF, [33](#)
- Ca_subset
 - oepdev::WavefunctionUnion, [71](#)
- Cb_subset
 - oepdev::WavefunctionUnion, [71](#)
- compute
 - oepdev::DIISManager, [36](#)
- compute_3D
 - oepdev::EETCouplingOEPotential, [37](#)
 - oepdev::ElectrostaticEnergyOEPotential, [38](#)
 - oepdev::OEPotential, [48](#)
 - oepdev::RepulsionEnergyOEPotential, [61](#)
- compute_benchmark
 - oepdev::ElectrostaticEnergySolver, [40](#)
 - oepdev::OEPDevSolver, [45](#)
 - oepdev::RepulsionEnergySolver, [64](#)
- compute_oep_based
 - oepdev::ElectrostaticEnergySolver, [40](#)
 - oepdev::OEPDevSolver, [45](#)
 - oepdev::RepulsionEnergySolver, [64](#)
- compute_shell
 - oepdev::AllAOShellCombinationsIterator, [32](#)
- create_superfunctional
 - oepdev, [25](#)
- DIISManager
 - oepdev::DIISManager, [36](#)
- ESPSolver
 - oepdev::ESPSolver, [43](#)
- extract_monomer
 - oepdev, [25](#)
- index
 - oepdev::AllAOIntegralsIterator, [30](#)
- OEPDevSolver
 - oepdev::OEPDevSolver, [44](#)
- OEPotential
 - oepdev::OEPotential, [47](#)
- OEPotential3D
 - oepdev::OEPotential3D, [50](#)
- oepdev, [23](#)
 - create_superfunctional, [25](#)
 - extract_monomer, [25](#)
 - psi, [26](#)
 - solve_scf, [25](#)
- oepdev::AllAOIntegralsIterator, [29](#)
 - AllAOIntegralsIterator, [30](#)
 - index, [30](#)
- oepdev::AllAOShellCombinationsIterator, [30](#)
 - AllAOShellCombinationsIterator, [31](#)
 - compute_shell, [32](#)
- oepdev::CPHF, [32](#)
 - CPHF, [33](#)
- oepdev::CubePoints3DIterator, [34](#)
- oepdev::CubePointsCollection3D, [35](#)
- oepdev::DIISManager, [35](#)
 - compute, [36](#)
 - DIISManager, [36](#)
 - put, [36](#)
 - update, [36](#)
- oepdev::EETCouplingOEPotential, [36](#)
 - compute_3D, [37](#)
- oepdev::ESPSolver, [42](#)
 - ESPSolver, [43](#)
- oepdev::ElectrostaticEnergyOEPotential, [37](#)
 - compute_3D, [38](#)
- oepdev::ElectrostaticEnergySolver, [38](#)
 - compute_benchmark, [40](#)
 - compute_oep_based, [40](#)
- oepdev::ElectrostaticPotential3D, [41](#)
- oepdev::OEPDevSolver, [43](#)
 - build, [44](#)
 - compute_benchmark, [45](#)
 - compute_oep_based, [45](#)
 - OEPDevSolver, [44](#)
- oepdev::OEPotential, [45](#)
 - build, [48](#)
 - compute_3D, [48](#)
 - OEPotential, [47](#)
 - write_cube, [48](#)
- oepdev::OEPotential3D
 - OEPotential3D, [50](#)
- oepdev::OEPotential3D < T >, [48](#)
- oepdev::Points3DIterator, [51](#)
 - build, [52](#), [54](#)

- Points3DIterator, 52
- oepdev::Points3DIterator::Point, 50
- oepdev::PointsCollection3D, 54
 - build, 56
- PointsCollection3D, 55
- oepdev::PotentialInt, 56
 - PotentialInt, 57, 58
 - set_charge_field, 58
- oepdev::RandomPoints3DIterator, 58
- oepdev::RandomPointsCollection3D, 60
- oepdev::RepulsionEnergyOEPotential, 60
 - compute_3D, 61
- oepdev::RepulsionEnergySolver, 61
 - compute_benchmark, 64
 - compute_oep_based, 64
- oepdev::ScalarField3D, 64
 - build, 66, 67
- oepdev::WavefunctionUnion, 67
 - Ca_subset, 71
 - Cb_subset, 71
 - WavefunctionUnion, 71
- Points3DIterator
 - oepdev::Points3DIterator, 52
- PointsCollection3D
 - oepdev::PointsCollection3D, 55
- PotentialInt
 - oepdev::PotentialInt, 57, 58
- psi, 25
 - oepdev, 26
 - read_options, 27
- put
 - oepdev::DIISManager, 36
- read_options
 - psi, 27
- set_charge_field
 - oepdev::PotentialInt, 58
- solve_scf
 - oepdev, 25
- update
 - oepdev::DIISManager, 36
- WavefunctionUnion
 - oepdev::WavefunctionUnion, 71
- write_cube
 - oepdev::OEPotential, 48