

oepdev 1.0.0-alpha

Novel methods for Quantum Chemistry of Extended Molecular Aggregates

Generated by Doxygen 1.8.6

Mon Mar 05 2018 12:46:02



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
<b>2</b>	<b>Introduction</b>	<b>3</b>
2.1	Research Project Methodology . . . . .	3
2.2	Expected Impact on the Development of Science, Civilization and Society . . . . .	4
2.3	The OEPDev Code . . . . .	4
<b>3</b>	<b>OEP Design.</b>	<b>5</b>
3.1	OEP Classes . . . . .	5
3.1.1	Structure of possible OEP-based expressions and their unification . . . . .	5
<b>4</b>	<b>List of One-Electron Potentials</b>	<b>7</b>
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	Charge-Transfer Energy OEP's . . . . .	8
4.4	Excitonic Energy Transfer OEP's . . . . .	8
4.4.1	ET contributions. . . . .	8
4.4.2	HT contributions. . . . .	8
4.4.3	CT contributions. . . . .	8
4.5	Full HF Interaction OEP's . . . . .	9
<b>5</b>	<b>Density-fitting Specialized for OEP's</b>	<b>11</b>
5.1	Fitting in Complete Space . . . . .	11
5.2	Fitting in Incomplete Space . . . . .	12
<b>6</b>	<b>Implemented Models</b>	<b>13</b>
6.1	Target Properties . . . . .	13
6.2	Target, Benchmark and Competing Models . . . . .	13
<b>7</b>	<b>Contributing to oep-dev</b>	<b>15</b>
7.1	Main Routine . . . . .	15
7.2	Header files in libraries . . . . .	15

7.3	Environmental variables	16
7.4	Documenting the code	16
7.5	Naming conventions	16
7.6	Track timing when evaluating the code	17
7.7	Use Object-Oriented Programming	17
<b>8</b>	<b>Advanced Usage</b>	<b>19</b>
8.1	OEPDev Code Structure	19
8.1.1	Main Routine	19
8.1.2	Modules	20
8.2	OEPDev Classes: Overview	20
8.2.1	OEP Module	20
8.2.1.1	OEPPotential	20
8.2.1.2	GeneralizedDensityFit	21
8.2.2	GEFP Module	21
8.2.2.1	GenEffPar	21
8.2.2.2	GenEffParFactory	21
8.2.2.3	GenEffFrag	21
8.2.3	OEPDev Solver Module	21
8.2.3.1	OEPDevSolver	21
8.3	Developing OEP's	21
8.3.1	Drafting an OEP Subclass	21
8.3.1.1	Implementing OEP Types	22
8.3.1.2	Abstract Base	23
<b>9</b>	<b>License</b>	<b>25</b>
<b>10</b>	<b>Module Index</b>	<b>27</b>
10.1	Modules	27
<b>11</b>	<b>Namespace Index</b>	<b>29</b>
11.1	Namespace List	29
<b>12</b>	<b>Hierarchical Index</b>	<b>31</b>
12.1	Class Hierarchy	31
<b>13</b>	<b>Class Index</b>	<b>33</b>
13.1	Class List	33
<b>14</b>	<b>File Index</b>	<b>37</b>
14.1	File List	37
<b>15</b>	<b>Module Documentation</b>	<b>39</b>

15.1	The Generalized One-Electron Potentials Library	39
15.1.1	Detailed Description	39
15.2	The OEPDev Solver Library	40
15.2.1	Detailed Description	40
15.3	The Generalized Effective Fragment Potentials Library	41
15.3.1	Detailed Description	41
15.4	The Integral Package Library	42
15.4.1	Detailed Description	43
15.4.2	Hermite Operators	43
15.4.2.1	Polynomial Expansions as Hermite Series	44
15.4.3	One-Body Integrals over Hermite Functions	45
15.4.4	Two-Body Integrals over Hermite Functions	45
15.4.5	The R(N,L,M) Coefficients	45
15.4.6	Function Documentation	46
15.4.6.1	d_N_n1_n2	46
15.4.6.2	make_mdh_D1_coeff	46
15.4.6.3	make_mdh_D2_coeff	46
15.4.6.4	make_mdh_D2_coeff_explicit_recursion	47
15.4.6.5	make_mdh_D3_coeff	47
15.4.6.6	make_mdh_R_coeff	48
15.5	The Three-Dimensional Vector Fields Library	49
15.5.1	Detailed Description	50
15.5.2	Function Documentation	50
15.5.2.1	OEPotential3D	50
15.5.2.2	OEPotential3D	50
15.6	The Density Functional Theory Library	52
15.7	The OEPDev Utilities	53
15.7.1	Detailed Description	54
15.7.2	Function Documentation	54
15.7.2.1	average_moment	54
15.7.2.2	create_superfunctional	55
15.7.2.3	extract_monomer	55
15.7.2.4	solve_scf	55
15.8	The OEPDev Testing Platform Library	57
15.8.1	Detailed Description	57
<b>16</b>	<b>Namespace Documentation</b>	<b>59</b>
16.1	oepdev Namespace Reference	59
16.1.1	Detailed Description	63
16.2	psi Namespace Reference	63

16.2.1 Detailed Description . . . . .	63
16.2.2 Function Documentation . . . . .	63
16.2.2.1 oepdev . . . . .	63
16.2.2.2 read_options . . . . .	64
<b>17 Class Documentation</b>	<b>65</b>
17.1 oepdev::ABCD Struct Reference . . . . .	65
17.1.1 Detailed Description . . . . .	65
17.2 oepdev::AbInitioPolarGEFactory Class Reference . . . . .	65
17.2.1 Detailed Description . . . . .	66
17.3 oepdev::AllAOIntegralsIterator_2 Class Reference . . . . .	66
17.3.1 Detailed Description . . . . .	67
17.3.2 Constructor & Destructor Documentation . . . . .	67
17.3.2.1 AllAOIntegralsIterator_2 . . . . .	67
17.3.2.2 AllAOIntegralsIterator_2 . . . . .	67
17.3.3 Member Function Documentation . . . . .	67
17.3.3.1 index . . . . .	67
17.4 oepdev::AllAOIntegralsIterator_4 Class Reference . . . . .	68
17.4.1 Detailed Description . . . . .	68
17.4.2 Constructor & Destructor Documentation . . . . .	68
17.4.2.1 AllAOIntegralsIterator_4 . . . . .	68
17.4.2.2 AllAOIntegralsIterator_4 . . . . .	69
17.4.3 Member Function Documentation . . . . .	69
17.4.3.1 index . . . . .	69
17.5 oepdev::AllAOShellCombinationsIterator_2 Class Reference . . . . .	69
17.5.1 Detailed Description . . . . .	70
17.5.2 Constructor & Destructor Documentation . . . . .	70
17.5.2.1 AllAOShellCombinationsIterator_2 . . . . .	70
17.5.2.2 AllAOShellCombinationsIterator_2 . . . . .	70
17.5.2.3 AllAOShellCombinationsIterator_2 . . . . .	70
17.5.2.4 AllAOShellCombinationsIterator_2 . . . . .	70
17.5.2.5 AllAOShellCombinationsIterator_2 . . . . .	71
17.5.3 Member Function Documentation . . . . .	71
17.5.3.1 compute_shell . . . . .	71
17.6 oepdev::AllAOShellCombinationsIterator_4 Class Reference . . . . .	71
17.6.1 Detailed Description . . . . .	72
17.6.2 Constructor & Destructor Documentation . . . . .	72
17.6.2.1 AllAOShellCombinationsIterator_4 . . . . .	72
17.6.2.2 AllAOShellCombinationsIterator_4 . . . . .	72
17.6.2.3 AllAOShellCombinationsIterator_4 . . . . .	72

17.6.2.4	AllAOShellCombinationsIterator_4	72
17.6.2.5	AllAOShellCombinationsIterator_4	73
17.6.3	Member Function Documentation	73
17.6.3.1	compute_shell	73
17.7	oepdev::AOIntegralsIterator Class Reference	73
17.7.1	Detailed Description	74
17.7.2	Member Function Documentation	74
17.7.2.1	build	74
17.7.2.2	build	74
17.8	oepdev::CAMM Class Reference	75
17.8.1	Detailed Description	75
17.9	oepdev::ChargeTransferEnergyOEPotential Class Reference	75
17.9.1	Detailed Description	76
17.10	oepdev::ChargeTransferEnergySolver Class Reference	76
17.10.1	Detailed Description	77
17.10.2	Member Function Documentation	78
17.10.2.1	compute_benchmark	78
17.10.2.2	compute_oep_based	79
17.11	oepdev::CPHF Class Reference	79
17.11.1	Detailed Description	81
17.11.2	Constructor & Destructor Documentation	82
17.11.2.1	CPHF	82
17.12	oepdev::CubePoints3DIterator Class Reference	83
17.12.1	Detailed Description	83
17.13	oepdev::CubePointsCollection3D Class Reference	84
17.13.1	Detailed Description	84
17.14	oepdev::DIISManager Class Reference	84
17.14.1	Detailed Description	85
17.14.2	Constructor & Destructor Documentation	85
17.14.2.1	DIISManager	85
17.14.3	Member Function Documentation	85
17.14.3.1	compute	85
17.14.3.2	put	85
17.14.3.3	update	85
17.15	oepdev::DMTPole Class Reference	86
17.15.1	Detailed Description	89
17.15.2	Constructor & Destructor Documentation	89
17.15.2.1	DMTPole	89
17.15.3	Member Function Documentation	89
17.15.3.1	build	89

17.15.3.2 energy . . . . .	89
17.15.3.3 potential . . . . .	90
17.16oepdev::DoubleGeneralizedDensityFit Class Reference . . . . .	90
17.16.1 Detailed Description . . . . .	91
17.16.2 Determination of the OEP matrix . . . . .	91
17.16.2.1 Theory behind the double GDF scheme . . . . .	92
17.16.3 Member Function Documentation . . . . .	92
17.16.3.1 compute . . . . .	92
17.17oepdev::EETCouplingOEPotential Class Reference . . . . .	92
17.17.1 Detailed Description . . . . .	93
17.18oepdev::ElectrostaticEnergyOEPotential Class Reference . . . . .	93
17.18.1 Detailed Description . . . . .	94
17.19oepdev::ElectrostaticEnergySolver Class Reference . . . . .	94
17.19.1 Detailed Description . . . . .	95
17.19.2 Member Function Documentation . . . . .	96
17.19.2.1 compute_benchmark . . . . .	96
17.19.2.2 compute_oep_based . . . . .	97
17.20oepdev::ElectrostaticPotential3D Class Reference . . . . .	97
17.20.1 Detailed Description . . . . .	97
17.21oepdev::ERI_1_1 Class Reference . . . . .	98
17.21.1 Detailed Description . . . . .	99
17.21.2 Implementation . . . . .	99
17.22oepdev::ERI_2_2 Class Reference . . . . .	99
17.22.1 Detailed Description . . . . .	100
17.22.2 Implementation . . . . .	100
17.23oepdev::ERI_3_1 Class Reference . . . . .	101
17.23.1 Detailed Description . . . . .	101
17.23.2 Implementation . . . . .	102
17.24oepdev::ESPSolver Class Reference . . . . .	102
17.24.1 Detailed Description . . . . .	103
17.24.2 Constructor & Destructor Documentation . . . . .	104
17.24.2.1 ESPSolver . . . . .	104
17.24.2.2 ESPSolver . . . . .	104
17.25oepdev::FFAbInitioPolarGEFactory Class Reference . . . . .	104
17.25.1 Detailed Description . . . . .	104
17.26oepdev::Field3D Class Reference . . . . .	105
17.26.1 Detailed Description . . . . .	107
17.26.2 Constructor & Destructor Documentation . . . . .	107
17.26.2.1 Field3D . . . . .	107
17.26.3 Member Function Documentation . . . . .	107



17.26.3.1 build . . . . .	107
17.26.3.2 build . . . . .	108
17.27oepdev::Fourier9 Struct Reference . . . . .	108
17.27.1 Detailed Description . . . . .	108
17.28oepdev::GenEffFrag Class Reference . . . . .	109
17.28.1 Detailed Description . . . . .	110
17.28.2 Member Function Documentation . . . . .	110
17.28.2.1 susceptibility . . . . .	110
17.28.2.2 susceptibility . . . . .	110
17.28.2.3 susceptibility . . . . .	110
17.29oepdev::GenEffPar Class Reference . . . . .	111
17.29.1 Detailed Description . . . . .	113
17.29.2 Member Function Documentation . . . . .	113
17.29.2.1 allocate . . . . .	113
17.29.2.2 compute_density_matrix . . . . .	113
17.29.2.3 compute_density_matrix . . . . .	113
17.29.2.4 compute_density_matrix . . . . .	114
17.29.2.5 compute_density_matrix . . . . .	114
17.29.2.6 set_susceptibility . . . . .	114
17.29.2.7 susceptibility . . . . .	114
17.29.2.8 susceptibility . . . . .	115
17.29.2.9 susceptibility . . . . .	115
17.30oepdev::GenEffParFactory Class Reference . . . . .	116
17.30.1 Detailed Description . . . . .	117
17.30.2 Member Function Documentation . . . . .	117
17.30.2.1 build . . . . .	117
17.31oepdev::GeneralizedDensityFit Class Reference . . . . .	118
17.31.1 Detailed Description . . . . .	119
17.31.2 Member Function Documentation . . . . .	120
17.31.2.1 build . . . . .	120
17.31.2.2 build . . . . .	120
17.31.2.3 compute . . . . .	120
17.32oepdev::GeneralizedPolarGEFactory Class Reference . . . . .	120
17.32.1 Detailed Description . . . . .	123
17.33oepdev::IntegralFactory Class Reference . . . . .	125
17.33.1 Detailed Description . . . . .	126
17.34oepdev::LinearGradientNonUniformEFieldPolarGEFactory Class Reference . . . . .	126
17.34.1 Detailed Description . . . . .	127
17.35oepdev::LinearNonUniformEFieldPolarGEFactory Class Reference . . . . .	127
17.35.1 Detailed Description . . . . .	128

17.36	<a href="#">oepdev::LinearUniformEFieldPolarGEFactory Class Reference</a>	128
17.36.1	Detailed Description	129
17.37	<a href="#">oepdev::MultipoleConvergence Class Reference</a>	129
17.37.1	Detailed Description	130
17.38	<a href="#">oepdev::NonUniformEFieldPolarGEFactory Class Reference</a>	130
17.38.1	Detailed Description	130
17.39	<a href="#">oepdev::OEPDevSolver Class Reference</a>	130
17.39.1	Detailed Description	131
17.39.2	Constructor & Destructor Documentation	132
17.39.2.1	OEPDevSolver	132
17.39.3	Member Function Documentation	132
17.39.3.1	build	132
17.39.3.2	compute_benchmark	132
17.39.3.3	compute_oep_based	132
17.40	<a href="#">oepdev::OEPotential Class Reference</a>	133
17.40.1	Detailed Description	135
17.40.2	Constructor & Destructor Documentation	135
17.40.2.1	OEPotential	135
17.40.2.2	OEPotential	135
17.40.3	Member Function Documentation	135
17.40.3.1	build	135
17.40.3.2	build	135
17.40.3.3	make_oeps3d	136
17.41	<a href="#">oepdev::OEPotential3D&lt; T &gt; Class Template Reference</a>	136
17.41.1	Detailed Description	137
17.42	<a href="#">oepdev::OEPTyp Struct Reference</a>	137
17.42.1	Detailed Description	138
17.43	<a href="#">oepdev::PerturbCharges Struct Reference</a>	138
17.43.1	Detailed Description	138
17.44	<a href="#">oepdev::Points3Dlterator::Point Struct Reference</a>	138
17.45	<a href="#">oepdev::Points3Dlterator Class Reference</a>	139
17.45.1	Detailed Description	140
17.45.2	Constructor & Destructor Documentation	140
17.45.2.1	Points3Dlterator	140
17.45.3	Member Function Documentation	140
17.45.3.1	build	140
17.45.3.2	build	141
17.45.3.3	build	142
17.46	<a href="#">oepdev::PointsCollection3D Class Reference</a>	142
17.46.1	Detailed Description	143

17.46.2 Constructor & Destructor Documentation . . . . .	143
17.46.2.1 PointsCollection3D . . . . .	143
17.46.3 Member Function Documentation . . . . .	144
17.46.3.1 build . . . . .	144
17.46.3.2 build . . . . .	144
17.46.3.3 build . . . . .	144
17.47oepdev::PolarGEFactory Class Reference . . . . .	144
17.47.1 Detailed Description . . . . .	145
17.48oepdev::PotentialInt Class Reference . . . . .	146
17.48.1 Detailed Description . . . . .	146
17.48.2 Constructor & Destructor Documentation . . . . .	146
17.48.2.1 PotentialInt . . . . .	146
17.48.2.2 PotentialInt . . . . .	146
17.48.2.3 PotentialInt . . . . .	147
17.48.3 Member Function Documentation . . . . .	147
17.48.3.1 set_charge_field . . . . .	147
17.49oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory Class Reference . . . . .	147
17.49.1 Detailed Description . . . . .	148
17.50oepdev::QuadraticNonUniformEFieldPolarGEFactory Class Reference . . . . .	149
17.50.1 Detailed Description . . . . .	149
17.51oepdev::QuadraticUniformEFieldPolarGEFactory Class Reference . . . . .	149
17.51.1 Detailed Description . . . . .	150
17.52oepdev::RandomPoints3DIterator Class Reference . . . . .	150
17.52.1 Detailed Description . . . . .	151
17.53oepdev::RandomPointsCollection3D Class Reference . . . . .	152
17.53.1 Detailed Description . . . . .	152
17.54oepdev::RepulsionEnergyOEPotential Class Reference . . . . .	152
17.54.1 Detailed Description . . . . .	153
17.55oepdev::RepulsionEnergySolver Class Reference . . . . .	153
17.55.1 Detailed Description . . . . .	154
17.55.2 Member Function Documentation . . . . .	157
17.55.2.1 compute_benchmark . . . . .	157
17.55.2.2 compute_oep_based . . . . .	157
17.56oepdev::RHPerturbed Class Reference . . . . .	158
17.56.1 Detailed Description . . . . .	159
17.57oepdev::ShellCombinationsIterator Class Reference . . . . .	159
17.57.1 Detailed Description . . . . .	161
17.57.2 Constructor & Destructor Documentation . . . . .	161
17.57.2.1 ShellCombinationsIterator . . . . .	161
17.57.3 Member Function Documentation . . . . .	162

17.57.3.1	ao_iterator	162
17.57.3.2	build	162
17.57.3.3	build	162
17.57.3.4	compute_shell	162
17.58	oepdev::SingleGeneralizedDensityFit Class Reference	163
17.58.1	Detailed Description	163
17.58.2	Determination of the OEP matrix	163
17.58.3	Member Function Documentation	164
17.58.3.1	compute	164
17.59	oepdev::GeneralizedPolarGEFactory::StatisticalSet Struct Reference	164
17.59.1	Detailed Description	164
17.60	oepdev::test::Test Class Reference	165
17.60.1	Detailed Description	166
17.61	oepdev::TwoBodyAOInt Class Reference	166
17.61.1	Member Function Documentation	166
17.61.1.1	compute	166
17.61.1.2	compute	167
17.62	oepdev::TwoElectronInt Class Reference	167
17.62.1	Detailed Description	168
17.62.2	Member Function Documentation	169
17.62.2.1	compute_shell	169
17.63	oepdev::UniformEFieldPolarGEFactory Class Reference	169
17.63.1	Detailed Description	169
17.64	oepdev::UnitaryOptimizer Class Reference	170
17.64.1	Detailed Description	172
17.64.2	Constructor & Destructor Documentation	173
17.64.2.1	UnitaryOptimizer	173
17.64.2.2	UnitaryOptimizer	174
17.64.2.3	UnitaryOptimizer	174
17.65	oepdev::UnitaryOptimizer_4_2 Class Reference	174
17.65.1	Detailed Description	177
17.65.2	Constructor & Destructor Documentation	178
17.65.2.1	UnitaryOptimizer_4_2	178
17.65.2.2	UnitaryOptimizer_4_2	178
17.66	oepdev::UnitaryTransformedMOPolarGEFactory Class Reference	178
17.66.1	Detailed Description	179
17.67	oepdev::WavefunctionUnion Class Reference	179
17.67.1	Detailed Description	182
17.67.2	Constructor & Destructor Documentation	183
17.67.2.1	WavefunctionUnion	183

17.67.3 Member Function Documentation . . . . .	183
17.67.3.1 Ca_subset . . . . .	183
17.67.3.2 Cb_subset . . . . .	183
<b>18 File Documentation</b>	<b>185</b>
18.1 include/oepdev_files.h File Reference . . . . .	185
18.2 include/oepdev_options.h File Reference . . . . .	185
18.3 main.cc File Reference . . . . .	186
18.4 oepdev/lib3d/dmtp.h File Reference . . . . .	186
18.5 oepdev/lib3d/esp.h File Reference . . . . .	187
18.6 oepdev/libgefp/gefp.h File Reference . . . . .	187
18.7 oepdev/libints/eri.h File Reference . . . . .	188
18.8 oepdev/libints/recurr.h File Reference . . . . .	189
18.9 oepdev/liboep/oep.h File Reference . . . . .	190
18.10 oepdev/liboep/oep_gdf.h File Reference . . . . .	191
18.11 oepdev/libpsi/integral.h File Reference . . . . .	191
18.12 oepdev/libpsi/potential.h File Reference . . . . .	192
18.13 oepdev/libsolver/solver.h File Reference . . . . .	192
18.14 oepdev/libtest/test.h File Reference . . . . .	193
18.15 oepdev/libutil/diis.h File Reference . . . . .	193
18.16 oepdev/libutil/integrals_iter.h File Reference . . . . .	194
18.17 oepdev/libutil/scf_perturb.h File Reference . . . . .	194
18.18 oepdev/libutil/unitary_optimizer.h File Reference . . . . .	195
18.19 oepdev/libutil/util.h File Reference . . . . .	196
18.20 oepdev/libutil/wavefunction_union.h File Reference . . . . .	197
<b>19 Example Documentation</b>	<b>199</b>
19.1 example_cphf.cc . . . . .	199
19.2 example_integrals_iter.cc . . . . .	199
19.3 example_scf_perturb.cc . . . . .	200
<b>Index</b>	<b>201</b>



# Chapter 1

## Main Page

### oep-dev

Generalized One-Electron Potentials: Development Platform.

Contact: Bartosz Błasiak ([blasiak.bartosz@gmail.com](mailto: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](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](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](https://github.com/globulion/oepdev/blob/master/doc/git/doc_programming_etiquette.md) "Programming Etiquette"
- [Current Issues](#)

This wikipages will be updated soon.

### References

[1] [B. Błasiak, "One-Particle Density Matrix Polarization Susceptibility Tensors", \\*Submitted\\* \\*\\*2018\\*\\* XX, XXX](#)





## Chapter 2

# Introduction

Exploring biological phenomena at molecular scale is oftentimes indispensable to develop new drugs and intelligent materials.

Most of relevant system properties are affected by intermolecular interactions with nearby environment such as solvent or closely bound electronic chromophores. Studying such molecular aggregates requires rigorous and accurate quantum chemistry methods, the cost of which grows very fast with the number of electrons. Despite many methodologies have been devised to describe energetic and dynamical properties of **extended molecular systems** efficiently and accurately, there exist particularly difficult cases in which modelling is still challenging:

- describing electronic transitions in solution or
- when coupled with other electronic transition via resonance energy transfer,
- performing molecular dynamics at very high level of theory including dynamic electron correlation,
- vibrational frequency calculations of particular normal mode in condensed phases

and so on. The reason behind (sometimes prohibitively) high costs of fully *ab initio* calculations in the above areas is the complexity of mathematical models often based on wave functions rather than (conceptually more straightforward) electronic densities. On the other hand, it has been pointed out before that the one-electron density distributions are of particular importance in chemistry. It can be thus utilized as a means of developing a general model that re-expresses the physics of intermolecular interactions in terms of effective one-electron functions that are easier to handle in practice.

This Project will focus on finding a unified way to simplify various equations of Quantum Chemistry of extended molecular systems, i.e., molecular aggregates such as interacting chromophores and molecules solvated by water and other solvents. Indeed, one of the important difficulties encountered in Quantum Chemistry of large systems is the need of evaluation of special kind of numbers known as *electron repulsion integrals*, or in short, ERI's. In a typical calculation, the amount of ERI's can be as high as tens or even hundreds of millions (!) that unfortunately prevents from application of conventional methods when the number of particles in question is too large. In the Project, the complicated expressions involving ERI's shall be greatly simplified to reduce the computational costs as much as possible while introducing no or minor approximations to the original theories.

### 2.1 Research Project Methodology

In this Project the new theoretical protocol based on the one-electron effective potentials (OEP's) is developed. The main principle is to rewrite arbitrary sum of functions  $f$  of electron repulsion integrals (ERI's) by defining **OEP's** according to the following general prescription:

$$\begin{aligned}\sum_f f [(\phi_i^A \phi_j^A || \phi_k^B \phi_l^B)] &= (\phi_i^A | v_{kl}^B | \phi_j^A) \rightarrow \text{point charge or density fitting} \\ \sum_f f [(\phi_i^A \phi_j^B || \phi_k^B \phi_l^B)] &= (\phi_i^A | v_{kl}^B | \phi_j^B) \rightarrow \text{density fitting},\end{aligned}$$

where  $A$  and  $B$  denote different molecules and  $\phi_i$  is the  $i$ -th molecular orbital or basis function. Here,  $v_{kl}^B$  denotes the [List of One-Electron Potentials](#) *ab initio* "OEP matrix element". The technique described above will be applied to simplify expressions for

- short-range excitation energy transfer couplings between chlorophyll subunits of reaction centres in photosynthesis
- Pauli interaction repulsion energy
- charge-transfer interaction energy
- electric field-induced charge density polarization of molecules.

The above developments might be used in fragment-based *ab initio* molecular dynamics protocols of new generation.

## 2.2 Expected Impact on the Development of Science, Civilization and Society

The proposed OEP's are expected to significantly develop the fragment-based methods that are widely used in physical chemistry and modelling of biologically important systems. Owing to universality of OEP's, they could find applications in many branches of chemical science: non-empirical\* molecular dynamics, short-range resonance energy transfer in photosynthesis, electronic and vibrational solvatochromism, multidimensional spectroscopy and so on. In particular:

- the OEP-based models of Pauli repulsion energy and charge-transfer (CT) energy could be used to improve the computational performance of the second generation effective fragment potential method (EFP2). At present, the CT term is very time consuming and due to this reasons it is not used in most of applications of EFP2 to perform molecular dynamics simulations.
- the OEP-based model of EET couplings could significantly improve modelling of energy transfer in the light harvesting complexes. At present, short-range phenomena (Dexter mechanisms of EET) are very difficult to efficiently and quantitatively asses when performing statistical averaging and applying to large molecular aggregates. Such Dexter effects could be computed by using OEP's in much more efficient manner without loosing high accuracy of parent TDFI-TI method.
- the density matrix polarization (DMS) tensors could be used in new generation fragment-based *ab initio* molecular dynamics protocols that rigorously take into consideration electron correlation effects.

Therefore, it is strongly believed that the OEP's could have an indirect impact on the design of novel drugs and materials for industry.

## 2.3 The OEPDev Code

To pursue the above challenges in the field of computational quantum chemistry of extended molecular aggregates, the OEPDev platform is developed. Accurate and efficient *ab initio* [models](#) based on OEP's are implemented in the OEPDev code, along with the state-of-the-art benchmark and competing methods. Written entirely in C++, OEPDev is a plugin to Psi4 quantum chemistry package. Therefore, compilation and running the OEPDev code is straightforward and follows the API interface similar to the one used in Psi4 with just a few [specific programming conventions](#). The detailed discussion about using the OEPDev code can be found in [advanced usage section](#).

## 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
$(\mu \hat{v}^{A[\mu]} \sigma)$	$\sum_{I \in A} v_{\mu I}^A S_{I\sigma}$	$\sum_{\alpha \in A} q_{\alpha}^{A[\mu]} V_{\mu\sigma}^{(\alpha)}$
$(i \hat{v}^{A[i]} j)$	$\sum_{I \in A} v_{ii}^A S_{Ij}$	$\sum_{\alpha \in A} q_{\alpha}^{A[i]} V_{ij}^{(\alpha)}$

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

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.

### Note

Add here a table with all the OEP types along with their symbols used in the OEPDev code (e.g., `Murrell-Sl` etc).

## 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 Murrel'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 Charge-Transfer Energy OEP's

To be added here!

### 4.4 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.

#### 4.4.1 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:

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

#### 4.4.2 HT contributions.

Do be derived.

#### 4.4.3 CT contributions.

To be derived.

## 4.5 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.

### 5.1 Fitting in Complete Space

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

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

under the necessary assumption that the auxiliary basis set is *complete*. In a special case when the basis set is orthogonal (e.g., molecular orbitals) the above relation simplifies to

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

It can be easily shown that the above general and exact expansion can be obtained by performing a density fitting in the complete space. 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:

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

which we shall refer here as to the matrix form of the OEP operator. By constructing the least-squares objective function

$$Z[\{G_{\xi}^{(i)}\}] = \int d\mathbf{r}_1 \left[ v(\mathbf{r}_1) \phi_i(\mathbf{r}_1) - \sum_{\xi} G_{\xi}^{(i)} \varphi_{\xi}(\mathbf{r}_1) \right]^2$$

and requiring that

$$\frac{\partial Z[\{G_{\xi}^{(i)}\}]}{\partial G_{\mu}^{(i)}} = 0 \text{ for all } \mu$$

we find the coefficients  $G_{\xi}^{(i)}$  to be

$$\mathbf{G}^{(i)} = \mathbf{v}^{(i)} \cdot \mathbf{S}^{-1}$$

where

$$\begin{aligned} v_{\eta}^{(i)} &= (\eta|v|i) \\ S_{\eta\xi} &= (\eta|\xi) \end{aligned}$$

or explicitly

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

identical to what we obtained from application of the resolution of identity in space spanned by non-orthogonal complete set of basis vectors.

Since matrix elements of an OEP operator in auxiliary space can be computed in the same way as the matrix elements with any other basis function, one can formally write the following identity

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

where  $X$  is an arbitrary orbital. When the other orbital 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.

## 5.2 Fitting in Incomplete Space

Density fitting scheme from previous section has practical disadvantage of a nearly-complete basis set being usually very large (spanned by large amount of basis set vectors). Any non-complete basis set won't work in the previous example. Since most of basis sets used in quantum chemistry do not form a complete set, it is beneficial to design a modified scheme in which it is possible to obtain the **effective** matrix elements of the OEP operator in a **incomplete** auxiliary space. This can be achieved by minimizing the following objective function

$$Z[\{G_{\xi}^{(i)}\}] = \iint d\mathbf{r}_1 d\mathbf{r}_2 \frac{\left[ v(\mathbf{r}_1) \phi_i(\mathbf{r}_1) - \sum_{\xi} G_{\xi}^{(i)} \phi_{\xi}(\mathbf{r}_1) \right] \left[ v(\mathbf{r}_2) \phi_i(\mathbf{r}_2) - \sum_{\xi} G_{\xi}^{(i)} \phi_{\xi}(\mathbf{r}_2) \right]}{|\mathbf{r}_1 - \mathbf{r}_2|}$$

Thus requesting that

$$\frac{\partial Z[\{G_{\xi}^{(i)}\}]}{\partial G_{\mu}^{(i)}} = 0 \text{ for all } \mu$$

we find the coefficients  $G_{\xi}^{(i)}$  to be

$$\mathbf{G}^{(i)} = \mathbf{b}^{(i)} \cdot \mathbf{A}^{-1}$$

where

$$b_{\eta}^{(i)} = (\eta||vi)$$

$$A_{\eta\xi} = (\eta||\xi)$$

The symbol  $||$  is to denote the operator  $r_{12}^{-1}$  and double integration over  $\mathbf{r}_1$  and  $\mathbf{r}_2$ . Thus, it is clear that in order to use this generalized density fitting scheme one must to compute two-centre electron repulsion integrals (implemented in [oepdev::ERI\\_1\\_1](#)) as well as four-centre asymmetric electron repulsion integrals of the type  $(\alpha\beta\gamma||\eta)$  (implemented in [oepdev::ERI\\_3\\_1](#)).

## 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

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. Include files directly related to `main.cc` are stored in the `include` directory, where only header files are present. Options are specified in `include/oepdev_options.h` whereas macros and defines in `include/oepdev_files.h`. 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 Track timing when evaluating the code

It is useful to track time elapsed for performing a particular task by a computer. For this, use `psi::timer_on` and `psi::timer_off` functions defined in `psi4/libqt/qt.h`. Psi4 always generates the report file `timer.dat` that contains all the defined timings. For example,

```
#include "psi/libqt/qt.h"
psi::timer_on("OEP      E(Paul) Murrell-etal S1  ");
// Your code goes here
psi::timer_off("OEP      E(Paul) Murrell-etal S1  ");
```

To maintain the printout in a neat form, the timing associated with the OEPDev code should be generated via `misc/python/timing.py` utility script.

## 7.7 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

# Advanced Usage

This section is addressed for advanced users.

Make sure you have first read [the introduction](#) before proceeding.

### 8.1 OEPDev Code Structure

As a plugin to Psi4, OEPDev consists of the `main.cc` file with the plugin main routine, `include/oepdev_options.h` specifying the options of the plugin, `include/oepdev_files.h` defining all global macros and environmental variables, as well as the `oepdev` directory. The latter contains the actual OEPDev code that is divided into several subdirectories called `modules`.

#### 8.1.1 Main Routine

Before the actual OEPDev calculations are started, the wavefunction of the input molecular aggregate is computed by Psi4. See the plugin driver script `pymodule.py` for more details on how the calculation environment is initialized. Subsequently, one out of four types of target operations can be performed by the program:

1. `OEP_BUILD` - Compute the OEP effective parameters for one molecule.
2. `DMATPOL` - Compute the generalized density matrix susceptibility tensors (DMS's) for one molecule.
3. `SOLVER` - Perform calculations for a molecular aggregate. As for now, only dimers are handled.
4. `TEST` - Perform the testing routine.

The first two modes are single molecule calculations. `OEP_BUILD` uses the `oepdev::OEPotential::build` static factory to create OEP objects whereas `DMATPOL` uses the `oepdev::GenEffParFactory::build` static factory to create generalized effective fragment parameters (GEFP's) for polarization.

#### Note

In the future, `OEP_BUILD` will be handled also by `oepdev::GenEffParFactory::build` since OEP parameters are part of the GEFP's.

`SOLVER` requires at least molecular dimer and the `oepdev::WavefunctionUnion` object (being the Hartree product of the unperturbed monomer wavefunctions) is constructed at the beginning, which is then passed to the `oepdev::OEPDevSolver::build` static factory. `TEST` can refer to single- or multiple-molecule calculations, whereby each of the testing routines is listed in the `cmake/CTestTestfile.cmake.in` file.

## 8.1.2 Modules

The source code is distributed into directories called modules:

- `liboep`
- `libgefp`
- `libsolver`
- `libints`
- `libpsi`
- `lib3d`
- `libutil`
- `libtest`

See Modules for a detailed description of each of the modules.

## 8.2 OEPDev Classes: Overview

### 8.2.1 OEP Module

The OEP module located in `oepdev/liboep` consists of the following abstract bases:

- `oepdev::OEPotential` implementing the OEP,
- `oepdev::GeneralizedDensityFit` implementing the GDF technique.

Each of the bases contains static factory method called `build` that creates instances of chosen subclasses. The module contains also a structure `oepdev::OEType` which is a container storing all the data associated with a particular OEP: type name, dimensions, OEP coefficients and whether is density-fitted or not.

#### 8.2.1.1 OEPPotential

It is a container and computer class of OEP. Among others, the most important public method is `oepdev::OEPPotential::compute` which computes all the OEP's (by iterating over all possible OEP types within a chosen OEP subclass or category). OEP's can be extracted by `oepdev::OEPPotential::oep` method, for instance. From protected attributes, each OEPPotential instance stores blocks of the LCAO-MO matrices associated with the occupied (`cOcc_`) and virtual (`cVir_`) MO's. It also contains the pointers to the primary, auxiliary and intermediate basis sets (`primary_`, `auxiliary_` and `intermediate_`, accordingly). Usage example:

```
#include "oepdev/liboep/oep.h"
oep = oepdev::OEPPotential::build("ELECTROSTATIC ENERGY", wfn, options);
oep->compute();
oep->write_cube("V", "oep_cube_file");
```

So far, four OEPPotential subclasses are implemented, from which `oepdev::ElectrostaticEnergyOEPPotential` and `oepdev::RepulsionEnergyOEPPotential` are fully operative, while the rest is under development.

## 8.2.1.2 GeneralizedDensityFit

## 8.2.2 GEFP Module

## 8.2.2.1 GenEffPar

## 8.2.2.2 GenEffParFactory

## 8.2.2.3 GenEffFrag

## 8.2.3 OEPDev Solver Module

## 8.2.3.1 OEPDevSolver

## 8.3 Developing OEP's

OEP's are implemented in a suitable subclass of the `oepdev::OEPotential` base. Due to the fact that OEP's can be density-based or ESP-based, the classes `oepdev::GeneralizedDensityFit` as well as `oepdev::ESPSolver` are usually necessary in the implementations. Handling the one-electron integrals (OEI's) and the two-electron integrals (ERI's) in AO basis is implemented in `oepdev::IntegralFactory`. In particular, potential integrals evaluated at arbitrary centres can be accessed by using the `oepdev::PotentialInt` instances. Useful iterators for looping over AO ERI's the `oepdev::ShellCombinationsIterator` and `oepdev::AOIntegralsIterator` classes. Transformations of OEI's to MO basis can be easily achieved by transforming AO integral matrices by `cOcc_` and `cVir_` members of `OEPotential` instances, e.g., by using the `psi::Matrix::doublet` or `psi::Matrix::triplet` static methods. Transformations of ERI's to MO basis can be performed by using the `psi4/libtrans/integraltransform.h` library.

It is recommended that the implementation of all the new OEP's follows the following steps:

1. **Write the class framework.** This includes choosing a proper name of a `OEPotential` subclass, sketching the constructors and a destructor, and all the necessary methods.
2. **Implement OEP types.** Each type of OEP is implemented, including the 3D vector field in case ESP-based OEP's are of use.
3. **Update base factory method.** Add appropriate entries in the `oepdev::OEPotential::build` static factory method.

Below, we shall go through each of these steps separately and discuss them in detail.

## 8.3.1 Drafting an OEP Subclass

This stage is the design of the overall framework of OEP subclass. The name should end with `OEPotential` to maintain the convention used so far. The template for the header file definition can be depicted as follows:

```
class SampleOEPotential : public OEPotential
{
public:
    // Purely ESP-based OEP's
    SampleOEPotential(SharedWavefunction wfn, Options& options);

    // GDF-based OEP's
    SampleOEPotential(SharedWavefunction wfn, SharedBasisSet auxiliary, SharedBasisSet intermediate,
        Options& options);

    // Necessary destructor
    virtual ~SampleOEPotential();

    // Necessary computer
    virtual void compute(const std::string& oepType) override;

    // Necessary computer
```

```

virtual void compute_3D(const std::string& oepType,
                      const double& x, const double& y, const double& z, std::shared_ptr<psi::Vector>
                      & v) override;
// Necessary printer
virtual void print_header() const override;

private:
// Set defaults - good practice
void common_init();

// Auxiliary computers - exemplary
double compute_3D_sample_V(const double& x, const double& y, const double& z);
};

```

The constructors need to call the abstract base constructor and then specialized initializations. It is a good practice to put the specialized common initializers in a separate private method `common_init` (which is a convention in Psi4 and is adopted also in OEPDev). For instance, the exemplary constructor is show below:

```

SampleOEPotential::SampleOEPotential(SharedWavefunction wfn,
                                     SharedBasisSet auxiliary, SharedBasisSet intermediate, Options&
                                     options)
: OEPotential(wfn, auxiliary, intermediate, options)
{
    common_init();
}

void SampleOEPotential::common_init()
{
    int n1 = wfn->Ca_subset("AO", "OCC")->ncol();
    int n2 = auxiliary->nbf();
    int n3 = wfn->molecule()->natom();

    SharedMatrix mat_1 = std::make_shared<psi::Matrix>("G(S^{-1})", n2, n1);
    SharedMatrix mat_2 = std::make_shared<psi::Matrix>("G(S^{-2})", n3, n1);

    OEType type_1 = {"Murrell-etal.S1", true, n1, mat_1};
    OEType type_2 = {"Otto-Ladik.S2", false, n1, mat_2};

    oepTypes_[type_1.name] = type_1;
    oepTypes_[type_2.name] = type_2;
}

```

Note that the `oepdev::OEPotential::oepTypes_` attribute, which is a `std::map` of structures `oepdev::OEType`, is initialized here. All the OEP types need to be stated in the constructors. Destructors usually call nothing, unless dynamically allocated memory is also of use.

It is also a good practice to already sketch the `compute` method here by adding certain private computers, like in the example below:

```

void SampleOEPotential::compute(const std::string& oepType)
{
    if (oepType == "Murrell-etal.S1") this->compute_murrell_etal_s1(); // calls private method
    else if (oepType == "Otto-Ladik.S2") this->compute_otto_ladik_s2(); // calls private method
    else throw psi::PSIEXCEPTION("OEPDEV: Error. Incorrect OEP type specified!\n"); // for safety
}

void SampleOEPotential::compute_murrell_etal_s1()
{
    psi::timer_on ("OEP      E(Paul) Murrell-etal S1  ");
    /* Your implementation goes here */
    psi::timer_off("OEP      E(Paul) Murrell-etal S1  ");
}

```

### 8.3.1.1 Implementing OEP Types

Implementation of the inner body of `compute` method requires populating the members of `oepTypes_` with data. This means, that for each OEP type there has to be a specific implementation of OEP parameters. GDF-based OEP's need to create the `psi::Matrix` with OEP parameters and put them into `oepTypes_`. In the case of E-SP-based OEP's `compute_3D` method has to be additionally implemented before `compute` is fully functional. To implement `compute_3D`, `oepdev::OEPotential::make_oeps3d` method is of high relevance: it creates `oepdev::OEPotential3D<T>` instances, where `T` is the OEP subclass. These instances are `oepdev::Field3D` objects that define OEP's in 3D Euclidean space. For example,

```

void SampleOEPotential::compute_otto_ladik_s2()
{

```

```

// Switch on timer
psi::timer_on("OEP      E(Paul) Otto-Ladik S2      ");

// Create 3D field, automated through 'make_oeps3d'. Requires 'compute_3D' implementation.
std::shared_ptr<OEPotential3D<OEPotential>> oeps3d = this->make_oeps3d("Otto-Ladik.S2");
oeps3d->compute();

// Perform ESP fit to get OEP effective charges
ESPSolver esp(oeps3d);
esp.set_charge_sums(0.5);
esp.compute();

// Put the OEP coefficients into 'oepTypes_'
for (int i=0; i<esp.charges()->nrow(); ++i) {
    for (int o=0; o<oepTypes_["Otto-Ladik.S2"].n; ++o) {
        oepTypes_["Otto-Ladik.S2"].matrix->set(i, o, esp.charges()->get(i, o));
    }
}

// Switch off timer
psi::timer_off("OEP      E(Paul) Otto-Ladik S2      ");
}

// Necessary implementation for 'make_oeps3d' to work
void SampleOEPotential::compute_3D(const std::string& oepType, const double& x, const double& y, const
double& z, std::shared_ptr<psi::Vector>& v)
{
    // Loop over all possibilities for OEP types and exclude illegal names
    if (oepType == "Otto-Ladik.S2") {

        // this computes the actual values of OEP = v(x,y,z) and stores it in 'vec_otto_ladik_s2_'
        this->compute_3D_otto_ladik_s2(x, y, z);

        // Assign final value to the buffer vector
        for (int o = 0; o < oepTypes_["Otto-Ladik.S2"].n; ++o) v->set(o, vec_otto_ladik_s2_[o]);

    }
    else if (oepType == "Murrell-et.al.S1" ) { /* Even if it is not ESP-based OEP, this line is necessary */}
    else {
        throw psi::PSIEXCEPTION("OEPDEV: Error. Incorrect OEP type specified!\n"); // Safety
    }
}

```

Note that `make_oeps3d` is not overridable and is fully defined in the base. Do not call `oepdev::OEPotential3D` constructors in the `OEPotential` subclass (it can be done only from the level of the abstract base where all the pointers are dynamically converted to an appropriate data type due to polymorphism)!

### 8.3.1.2 Abstract Base



## **Chapter 9**

# **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 10

# Module Index

### 10.1 Modules

Here is a list of all modules:

The Generalized One-Electron Potentials Library . . . . .	39
The OEPDev Solver Library . . . . .	40
The Generalized Effective Fragment Potentials Library . . . . .	41
The Integral Package Library . . . . .	42
The Three-Dimensional Vector Fields Library . . . . .	49
The Density Functional Theory Library . . . . .	52
The OEPDev Utilities . . . . .	53
The OEPDev Testing Platform Library . . . . .	57



# Chapter 11

## Namespace Index

### 11.1 Namespace List

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

<a href="#">oepdev</a>	OEPPDev module namespace . . . . .	<a href="#">59</a>
<a href="#">psi</a>	Psi4 package namespace . . . . .	<a href="#">63</a>



## Chapter 12

# Hierarchical Index

### 12.1 Class Hierarchy

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

oepdev::ABCD . . . . .	65
oepdev::AOIntegralsIterator . . . . .	73
oepdev::AllAOIntegralsIterator_2 . . . . .	66
oepdev::AllAOIntegralsIterator_4 . . . . .	68
oepdev::CPHF . . . . .	79
CubicScalarGrid	
oepdev::CubePointsCollection3D . . . . .	84
oepdev::DIISManager . . . . .	84
enable_shared_from_this	
oepdev::DMTPole . . . . .	86
oepdev::CAMM . . . . .	75
oepdev::OEPDevSolver . . . . .	130
oepdev::ChargeTransferEnergySolver . . . . .	76
oepdev::ElectrostaticEnergySolver . . . . .	94
oepdev::RepulsionEnergySolver . . . . .	153
oepdev::OEPotential . . . . .	133
oepdev::ChargeTransferEnergyOEPotential . . . . .	75
oepdev::EETCouplingOEPotential . . . . .	92
oepdev::ElectrostaticEnergyOEPotential . . . . .	93
oepdev::RepulsionEnergyOEPotential . . . . .	152
oepdev::ESPSolver . . . . .	102
oepdev::Field3D . . . . .	105
oepdev::ElectrostaticPotential3D . . . . .	97
oepdev::OEPotential3D< T > . . . . .	136
oepdev::Fourier9 . . . . .	108
oepdev::GenEffFrag . . . . .	109
oepdev::GenEffPar . . . . .	111
oepdev::GenEffParFactory . . . . .	116
oepdev::PolarGEFactory . . . . .	144
oepdev::AbInitioPolarGEFactory . . . . .	65
oepdev::UnitaryTransformedMOPolarGEFactory . . . . .	178
oepdev::FFAbInitioPolarGEFactory . . . . .	104
oepdev::GeneralizedPolarGEFactory . . . . .	120
oepdev::NonUniformEFieldPolarGEFactory . . . . .	130
oepdev::LinearGradientNonUniformEFieldPolarGEFactory . . . . .	126
oepdev::LinearNonUniformEFieldPolarGEFactory . . . . .	127
oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory . . . . .	147

oepdev::QuadraticNonUniformEFieldPolarGEFactory . . . . .	149
oepdev::UniformEFieldPolarGEFactory . . . . .	169
oepdev::LinearUniformEFieldPolarGEFactory . . . . .	128
oepdev::QuadraticUniformEFieldPolarGEFactory . . . . .	149
oepdev::GeneralizedDensityFit . . . . .	118
oepdev::DoubleGeneralizedDensityFit . . . . .	90
oepdev::SingleGeneralizedDensityFit . . . . .	163
IntegralFactory	
oepdev::IntegralFactory . . . . .	125
oepdev::MultipoleConvergence . . . . .	129
oepdev::OEType . . . . .	137
oepdev::PerturbCharges . . . . .	138
oepdev::Points3DIterator::Point . . . . .	138
oepdev::Points3DIterator . . . . .	139
oepdev::CubePoints3DIterator . . . . .	83
oepdev::RandomPoints3DIterator . . . . .	150
oepdev::PointsCollection3D . . . . .	142
oepdev::CubePointsCollection3D . . . . .	84
oepdev::RandomPointsCollection3D . . . . .	152
PotentialInt	
oepdev::PotentialInt . . . . .	146
RHF	
oepdev::RHFPerturbed . . . . .	158
oepdev::ShellCombinationsIterator . . . . .	159
oepdev::AllAOShellCombinationsIterator_2 . . . . .	69
oepdev::AllAOShellCombinationsIterator_4 . . . . .	71
oepdev::GeneralizedPolarGEFactory::StatisticalSet . . . . .	164
oepdev::test::Test . . . . .	165
TwoBodyAOInt	
oepdev::TwoBodyAOInt . . . . .	166
oepdev::TwoElectronInt . . . . .	167
oepdev::ERI_1_1 . . . . .	98
oepdev::ERI_2_2 . . . . .	99
oepdev::ERI_3_1 . . . . .	101
oepdev::UnitaryOptimizer . . . . .	170
oepdev::UnitaryOptimizer_4_2 . . . . .	174
Wavefunction	
oepdev::WavefunctionUnion . . . . .	179

## Chapter 13

# Class Index

### 13.1 Class List

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

<a href="#">oepdev::ABCD</a>	Simple structure to hold the Fourier series expansion coefficients . . . . .	65
<a href="#">oepdev::AbInitioPolarGEFactory</a>	Polarization GEFP Factory from First Principles. Hartree-Fock Approximation . . . . .	65
<a href="#">oepdev::AllAOIntegralsIterator_2</a>	Loop over all possible ERI within a particular shell doublet . . . . .	66
<a href="#">oepdev::AllAOIntegralsIterator_4</a>	Loop over all possible ERI within a particular shell quartet . . . . .	68
<a href="#">oepdev::AllAOShellCombinationsIterator_2</a>	Loop over all possible ERI shells in a shell doublet . . . . .	69
<a href="#">oepdev::AllAOShellCombinationsIterator_4</a>	Loop over all possible ERI shells in a shell quartet . . . . .	71
<a href="#">oepdev::AOIntegralsIterator</a>	Iterator for AO Integrals. Abstract Base . . . . .	73
<a href="#">oepdev::CAMM</a>	Cumulative Atomic Multipole Moments . . . . .	75
<a href="#">oepdev::ChargeTransferEnergyOEPotential</a>	Generalized One-Electron Potential for Charge-Transfer Interaction Energy . . . . .	75
<a href="#">oepdev::ChargeTransferEnergySolver</a>	Compute the Charge-Transfer interaction energy between unperturbed wavefunctions . . . . .	76
<a href="#">oepdev::CPHF</a>	CPHF solver class . . . . .	79
<a href="#">oepdev::CubePoints3DIterator</a>	Iterator over a collection of points in 3D space. g09 Cube-like order . . . . .	83
<a href="#">oepdev::CubePointsCollection3D</a>	G09 cube-like ordered collection of points in 3D space . . . . .	84
<a href="#">oepdev::DIISManager</a>	DIIS manager . . . . .	84
<a href="#">oepdev::DMTPole</a>	Distributed Multipole Analysis Container and Computer. Abstract Base . . . . .	86
<a href="#">oepdev::DoubleGeneralizedDensityFit</a>	Generalized Density Fitting Scheme - Double Fit . . . . .	90
<a href="#">oepdev::EETCouplingOEPotential</a>	Generalized One-Electron Potential for EET coupling calculations . . . . .	92
<a href="#">oepdev::ElectrostaticEnergyOEPotential</a>	Generalized One-Electron Potential for Electrostatic Energy . . . . .	93
<a href="#">oepdev::ElectrostaticEnergySolver</a>	Compute the Coulombic interaction energy between unperturbed wavefunctions . . . . .	94

<a href="#">oepdev::ElectrostaticPotential3D</a>	
Electrostatic potential of a molecule . . . . .	97
<a href="#">oepdev::ERI_1_1</a>	
2-centre ERI of the form $(a O(2) b)$ where $O(2) = 1/r_{12}$ . . . . .	98
<a href="#">oepdev::ERI_2_2</a>	
4-centre ERI of the form $(ab O(2) cd)$ where $O(2) = 1/r_{12}$ . . . . .	99
<a href="#">oepdev::ERI_3_1</a>	
4-centre ERI of the form $(abc O(2) d)$ where $O(2) = 1/r_{12}$ . . . . .	101
<a href="#">oepdev::ESPSolver</a>	
Charges from Electrostatic Potential (ESP). A solver-type class . . . . .	102
<a href="#">oepdev::FFAbInitioPolarGEFactory</a>	
Polarization GEFP Factory from First Principles: Finite-Difference Model. Arbitrary level of theory . . . . .	104
<a href="#">oepdev::Field3D</a>	
General Vector Field in 3D Space. Abstract base . . . . .	105
<a href="#">oepdev::Fourier9</a>	
Simple structure to hold the Fourier series expansion coefficients for $N=4$ . . . . .	108
<a href="#">oepdev::GenEffFrag</a>	
Generalized Effective Fragment. Container Class . . . . .	109
<a href="#">oepdev::GenEffPar</a>	
Generalized Effective Fragment Parameters. Container Class . . . . .	111
<a href="#">oepdev::GenEffParFactory</a>	
Generalized Effective Fragment Factory. Abstract Base . . . . .	116
<a href="#">oepdev::GeneralizedDensityFit</a>	
Generalized Density Fitting Scheme. Abstract Base . . . . .	118
<a href="#">oepdev::GeneralizedPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	120
<a href="#">oepdev::IntegralFactory</a>	
Extended <a href="#">IntegralFactory</a> for computing integrals . . . . .	125
<a href="#">oepdev::LinearGradientNonUniformEFieldPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	126
<a href="#">oepdev::LinearNonUniformEFieldPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	127
<a href="#">oepdev::LinearUniformEFieldPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	128
<a href="#">oepdev::MultipoleConvergence</a>	
Multipole Convergence . . . . .	129
<a href="#">oepdev::NonUniformEFieldPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	130
<a href="#">oepdev::OEPDevSolver</a>	
Solver of properties of molecular aggregates. Abstract base . . . . .	130
<a href="#">oepdev::OEPotential</a>	
Generalized One-Electron Potential: Abstract base . . . . .	133
<a href="#">oepdev::OEPotential3D&lt; T &gt;</a>	
Class template for OEP 3D fields . . . . .	136
<a href="#">oepdev::OEType</a>	
Container to handle the type of One-Electron Potentials . . . . .	137
<a href="#">oepdev::PerturbCharges</a>	
Structure to hold perturbing charges . . . . .	138
<a href="#">oepdev::Points3DIterator::Point</a>	
Points3DIterator . . . . .	138
<a href="#">oepdev::Points3DIterator</a>	
Iterator over a collection of points in 3D space. Abstract base . . . . .	139
<a href="#">oepdev::PointsCollection3D</a>	
Collection of points in 3D space. Abstract base . . . . .	142
<a href="#">oepdev::PolarGEFactory</a>	
Polarization GEFP Factory. Abstract Base . . . . .	144
<a href="#">oepdev::PotentialInt</a>	
Computes potential integrals . . . . .	146



<a href="#">oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	147
<a href="#">oepdev::QuadraticNonUniformEFieldPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	149
<a href="#">oepdev::QuadraticUniformEFieldPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	149
<a href="#">oepdev::RandomPoints3DIterator</a>	
Iterator over a collection of points in 3D space. Random collection . . . . .	150
<a href="#">oepdev::RandomPointsCollection3D</a>	
Collection of random points in 3D space . . . . .	152
<a href="#">oepdev::RepulsionEnergyOEPotential</a>	
Generalized One-Electron Potential for Pauli Repulsion Energy . . . . .	152
<a href="#">oepdev::RepulsionEnergySolver</a>	
Compute the Pauli-Repulsion interaction energy between unperturbed wavefunctions . . . . .	153
<a href="#">oepdev::RHFPerturbed</a>	
RHF theory under electrostatic perturbation . . . . .	158
<a href="#">oepdev::ShellCombinationsIterator</a>	
Iterator for Shell Combinations. Abstract Base . . . . .	159
<a href="#">oepdev::SingleGeneralizedDensityFit</a>	
Generalized Density Fitting Scheme - Single Fit . . . . .	163
<a href="#">oepdev::GeneralizedPolarGEFactory::StatisticalSet</a>	
A structure to handle statistical data . . . . .	164
<a href="#">oepdev::test::Test</a>	
Manages test routines . . . . .	165
<a href="#">oepdev::TwoBodyAOInt</a>	
General Two Electron Integral . . . . .	166
<a href="#">oepdev::TwoElectronInt</a>	
General Two Electron Integral . . . . .	167
<a href="#">oepdev::UniformEFieldPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	169
<a href="#">oepdev::UnitaryOptimizer</a>	
Find the optimum unitary matrix of quadratic matrix equation . . . . .	170
<a href="#">oepdev::UnitaryOptimizer_4_2</a>	
Find the optimum unitary matrix for quartic-quadratic matrix equation with trace . . . . .	174
<a href="#">oepdev::UnitaryTransformedMOPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Scaling of MO Space . . . . .	178
<a href="#">oepdev::WavefunctionUnion</a>	
Union of two Wavefunction objects . . . . .	179



## Chapter 14

# File Index

### 14.1 File List

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

<a href="#">main.cc</a>	186
<a href="#">include/oepdev_files.h</a>	185
<a href="#">include/oepdev_options.h</a>	185
<a href="#">include/doxygen/oepdev_manual.h</a>	??
<a href="#">include/doxygen/oepdev_modules.h</a>	??
<a href="#">include/doxygen/oepdev_namespaces.h</a>	??
<a href="#">oepdev/lib3d/dmtp.h</a>	186
<a href="#">oepdev/lib3d/esp.h</a>	187
<a href="#">oepdev/lib3d/space3d.h</a>	??
<a href="#">oepdev/libgefp/gefp.h</a>	187
<a href="#">oepdev/libints/eri.h</a>	188
<a href="#">oepdev/libints/recur.h</a>	189
<a href="#">oepdev/liboep/oep.h</a>	190
<a href="#">oepdev/liboep/oep_gdf.h</a>	191
<a href="#">oepdev/libpsi/integral.h</a>	191
<a href="#">oepdev/libpsi/potential.h</a>	192
<a href="#">oepdev/libsolver/solver.h</a>	192
<a href="#">oepdev/libtest/test.h</a>	193
<a href="#">oepdev/libutil/cphf.h</a>	??
<a href="#">oepdev/libutil/diis.h</a>	193
<a href="#">oepdev/libutil/integrals_iter.h</a>	194
<a href="#">oepdev/libutil/scf_perturb.h</a>	194
<a href="#">oepdev/libutil/unitary_optimizer.h</a>	195
<a href="#">oepdev/libutil/util.h</a>	196
<a href="#">oepdev/libutil/wavefunction_union.h</a>	197



# Chapter 15

## Module Documentation

### 15.1 The Generalized One-Electron Potentials Library

Implements the goal of this project: The Generalized One-Electron Potentials (OEP's). You will find here OEP's for computation of Pauli repulsion energy, charge-transfer energy and others. The routines for the generalized density fitting are also implemented here. Located at `oepdev/liboep`.

#### Classes

- struct `oepdev::OEType`  
*Container to handle the type of One-Electron Potentials.*
- class `oepdev::OEPotential`  
*Generalized One-Electron Potential: Abstract base.*
- class `oepdev::ElectrostaticEnergyOEPotential`  
*Generalized One-Electron Potential for Electrostatic Energy.*
- class `oepdev::RepulsionEnergyOEPotential`  
*Generalized One-Electron Potential for Pauli Repulsion Energy.*
- class `oepdev::ChargeTransferEnergyOEPotential`  
*Generalized One-Electron Potential for Charge-Transfer Interaction Energy.*
- class `oepdev::EETCouplingOEPotential`  
*Generalized One-Electron Potential for EET coupling calculations.*
- class `oepdev::GeneralizedDensityFit`  
*Generalized Density Fitting Scheme. Abstract Base.*
- class `oepdev::SingleGeneralizedDensityFit`  
*Generalized Density Fitting Scheme - Single Fit.*
- class `oepdev::DoubleGeneralizedDensityFit`  
*Generalized Density Fitting Scheme - Double Fit.*

#### 15.1.1 Detailed Description

Implements the goal of this project: The Generalized One-Electron Potentials (OEP's). You will find here OEP's for computation of Pauli repulsion energy, charge-transfer energy and others. The routines for the generalized density fitting are also implemented here. Located at `oepdev/liboep`.

## 15.2 The OEPDev Solver Library

Implementations of various solvers for molecular properties as a functions of unperturbed monomeric wavefunctions. This is the place all target OEP-based models are implemented and compared with benchmark and competitor models. Located at `oepdev/libsolver`.

### Classes

- class `oepdev::OEPDevSolver`  
*Solver of properties of molecular aggregates. Abstract base.*
- class `oepdev::ElectrostaticEnergySolver`  
*Compute the Coulombic interaction energy between unperturbed wavefunctions.*
- class `oepdev::RepulsionEnergySolver`  
*Compute the Pauli-Repulsion interaction energy between unperturbed wavefunctions.*
- class `oepdev::ChargeTransferEnergySolver`  
*Compute the Charge-Transfer interaction energy between unperturbed wavefunctions.*

### 15.2.1 Detailed Description

Implementations of various solvers for molecular properties as a functions of unperturbed monomeric wavefunctions. This is the place all target OEP-based models are implemented and compared with benchmark and competitor models. Located at `oepdev/libsolver`.

## 15.3 The Generalized Effective Fragment Potentials Library

Implements the GEFP method, the far goal of the OEPDev project. Here you will find the containers for GEFP parameters, the density matrix susceptibility tensors and GEFP solvers. Located at `oepdev/libgefp`.

### Classes

- class `oepdev::GenEffPar`  
*Generalized Effective Fragment Parameters. Container Class.*
- class `oepdev::GenEffFrag`  
*Generalized Effective Fragment. Container Class.*
- class `oepdev::GenEffParFactory`  
*Generalized Effective Fragment Factory. Abstract Base.*
- class `oepdev::PolarGEFactory`  
*Polarization GEFP Factory. Abstract Base.*
- class `oepdev::AbInitioPolarGEFactory`  
*Polarization GEFP Factory from First Principles. Hartree-Fock Approximation.*
- class `oepdev::FFAbInitioPolarGEFactory`  
*Polarization GEFP Factory from First Principles: Finite-Difference Model. Arbitrary level of theory.*
- class `oepdev::GeneralizedPolarGEFactory`  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class `oepdev::UniformEFieldPolarGEFactory`  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class `oepdev::NonUniformEFieldPolarGEFactory`  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class `oepdev::LinearUniformEFieldPolarGEFactory`  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class `oepdev::QuadraticUniformEFieldPolarGEFactory`  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class `oepdev::LinearNonUniformEFieldPolarGEFactory`  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class `oepdev::QuadraticNonUniformEFieldPolarGEFactory`  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class `oepdev::LinearGradientNonUniformEFieldPolarGEFactory`  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class `oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory`  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class `oepdev::UnitaryTransformedMOPolarGEFactory`  
*Polarization GEFP Factory with Least-Squares Scaling of MO Space.*

### 15.3.1 Detailed Description

Implements the GEFP method, the far goal of the OEPDev project. Here you will find the containers for GEFP parameters, the density matrix susceptibility tensors and GEFP solvers. Located at `oepdev/libgefp`.

## 15.4 The Integral Package Library

Implementations of various two-, three- or four-centre two-body electron repulsion integrals via utilizing the McMurchie-Davidson recurrence scheme. Located at `oepdev/libints` and `oepdev/libpsi`.

### Classes

- class `oepdev::TwoElectronInt`  
*General Two Electron Integral.*
- class `oepdev::ERI_1_1`  
*2-centre ERI of the form  $\langle a|O(2)|b\rangle$  where  $O(2) = 1/r^{12}$ .*
- class `oepdev::ERI_2_2`  
*4-centre ERI of the form  $\langle ab|O(2)|cd\rangle$  where  $O(2) = 1/r^{12}$ .*
- class `oepdev::ERI_3_1`  
*4-centre ERI of the form  $\langle abc|O(2)|d\rangle$  where  $O(2) = 1/r^{12}$ .*
- class `oepdev::TwoBodyAOInt`
- class `oepdev::IntegralFactory`  
*Extended [IntegralFactory](#) for computing integrals.*
- class `oepdev::PotentialInt`  
*Computes potential integrals.*

### Macros

- `#define D1_INDEX(x, i, n) ((81*(x))+(9*(i))+(n))`  
*Get the index of McMurchie-Davidson-Hermite D1 coefficient stored in the `mdh_buffer_`, that is attributed to the x Cartesian coordinate from angular momentum i of function 1, and the Hermite index n.*
- `#define D2_INDEX(x, i, j, n) ((1377*(x))+(153*(i))+(17*(j))+(n))`  
*Get the index of McMurchie-Davidson-Hermite D2 coefficient stored in the `mdh_buffer_`, that is attributed to the x Cartesian coordinate from angular momenta i, j of function 1 and 2, and the Hermite index n.*
- `#define D3_INDEX(x, i, j, k, n) ((18225*(x))+(2025*(i))+(225*(j))+(25*(k))+(n))`  
*Get the index of McMurchie-Davidson-Hermite D3 coefficient stored in the `mdh_buffer_`, that is attributed to the x Cartesian coordinate from angular momenta i, j and k of function 1, 2 and 3, and the Hermite index n.*
- `#define R_INDEX(n, l, m, j) ((14739*(n))+(867*(l))+(51*(m))+(j))`  
*Get the index of McMurchie-Davidson R coefficient stored in the `mdh_buffer_R_` from angular momenta n, l and m and the Boys index j.*

### Functions

- double `oepdev::d_N_n1_n2` (int N, int n1, int n2, double PA, double PB, double aP)  
*Compute McMurchie-Davidson-Hermite (MDH) coefficient for binomial expansion.*
- void `oepdev::make_mdh_D1_coeff` (int n1, double aPd, double \*buffer)  
*Compute the McMurchie-Davidson-Hermite coefficients for monomial expansion.*
- void `oepdev::make_mdh_D2_coeff` (int n1, int n2, double aPd, double \*PA, double \*PB, double \*buffer)  
*Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion.*
- void `oepdev::make_mdh_D3_coeff` (int n1, int n2, int n3, double aPd, double \*PA, double \*PB, double \*PC, double \*buffer)  
*Compute the McMurchie-Davidson-Hermite coefficients for trinomial expansion.*
- void `oepdev::make_mdh_D2_coeff_explicit_recursion` (int n1, int n2, double aP, double \*PA, double \*PB, double \*buffer)  
*Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion by explicit recursion. This function makes the same changes to buffers as `oepdev::make_mdh_D2_coeff`, but implements it through explicit recursion by calls to `oepdev::d_N_n1_n2`. Therefore, it is slightly slower. Here for debugging purposes.*



- void `oepdev::make_mdh_R_coeff` (int N, int L, int M, double alpha, double a, double b, double c, double \*F, double \*buffer)

*Compute the McMurchie-Davidson R coefficients.*

### 15.4.1 Detailed Description

Implementations of various two-, three- or four-centre two-body electron repulsion integrals via utilizing the McMurchie-Davidson recurrence scheme. Located at `oepdev/libints` and `oepdev/libpsi`. Here, we define the primitive Gaussian type functions (GTO's)

$$\begin{aligned}\phi_i(\mathbf{r}) &\equiv x_A^{n_1} y_A^{l_1} z_A^{m_1} e^{-\alpha_1 r_A^2} \\ \phi_j(\mathbf{r}) &\equiv x_B^{n_2} y_B^{l_2} z_B^{m_2} e^{-\alpha_2 r_B^2} \\ \phi_k(\mathbf{r}) &\equiv x_C^{n_3} y_C^{l_3} z_C^{m_3} e^{-\alpha_3 r_C^2}\end{aligned}$$

in which  $\mathbf{r}_A \equiv \mathbf{r} - \mathbf{A}$  and so on.  $\mathbf{A}$  is the centre of the GTO,  $\alpha_1$  its exponent, whereas  $n_1, l_1, m_1$  the Cartesian angular momenta, with the total angular momentum  $\theta_1 = n_1 + l_1 + m_1$ .

In OEPDev implementations, the following definition shall be in use:

$$\begin{aligned}\mathbf{P} &\equiv \frac{\alpha_1 \mathbf{A} + \alpha_2 \mathbf{B}}{\alpha_1 + \alpha_2} \\ \mathbf{Q} &\equiv \frac{\alpha_3 \mathbf{C} + \alpha_4 \mathbf{D}}{\alpha_3 + \alpha_4} \\ \mathbf{R} &\equiv \frac{\alpha_1 \mathbf{A} + \alpha_2 \mathbf{B} + \alpha_3 \mathbf{C}}{\alpha_1 + \alpha_2 + \alpha_3} \\ \alpha_P &\equiv \alpha_1 + \alpha_2 \\ \alpha_Q &\equiv \alpha_3 + \alpha_4 \\ \alpha_R &\equiv \alpha_1 + \alpha_2 + \alpha_3\end{aligned}$$

The unnormalized products of primitive GTO's are denoted here as

$$\begin{aligned}[ij] &\equiv \phi_i(\mathbf{r}) \phi_j(\mathbf{r}) \\ [ijk] &\equiv \phi_i(\mathbf{r}) \phi_j(\mathbf{r}) \phi_k(\mathbf{r})\end{aligned}$$

### 15.4.2 Hermite Operators

It is convenient to define

$$\Lambda_j(x_P; \alpha_P) \equiv \left( \frac{\partial}{\partial P_x} \right)^j = \alpha_P^{j/2} H_j(\sqrt{\alpha_P} x_P)$$

where  $H_j(x)$  is the Hermite polynomial of order  $j$  evaluated at  $x$ . Introduction of the above Hermite operator can be used by invoking the recurrence relationship due to Hermite polynomial properties:

$$x_A \Lambda_j(x_P; \alpha_P) = j \Lambda_{j-1} + |\mathbf{P} - \mathbf{A}|_x \Lambda_j + \frac{1}{2\alpha_P} \Lambda_{j+1}$$

This can be directly used to derive very useful McMurchie-Davidson-Hermite coefficients as expansion coefficients of the polynomial expansions.

### 15.4.2.1 Polynomial Expansions as Hermite Series

By using the previous relation, it is possible to express the following expansions in Hermite series:

$$\begin{aligned} x_A^{n_1} &= \sum_{N=0}^{n_1} d_N^{n_1} \Lambda_N(x_A; \alpha_A) \\ x_A^{n_1} x_B^{n_2} &= \sum_{N=0}^{n_1+n_2} d_N^{n_1 n_2} \Lambda_N(x_P; \alpha_P) \\ x_A^{n_1} x_B^{n_2} x_C^{n_3} &= \sum_{N=0}^{n_1+n_2+n_3} d_N^{n_1 n_2 n_3} \Lambda_N(x_R; \alpha_R) \end{aligned}$$

The recurrence relationships can be easily found and they read

$$d_N^{n_1+1} = \frac{1}{2\alpha_A} d_{N-1}^{n_1} + (N+1) d_{N+1}^{n_1}$$

as well as

$$\begin{aligned} d_N^{n_1+1, n_2} &= \frac{1}{2\alpha_P} d_{N-1}^{n_1 n_2} + |\mathbf{P} - \mathbf{A}|_x d_N^{n_1 n_2} + (N+1) d_{N+1}^{n_1 n_2} \\ d_N^{n_1, n_2+1} &= \frac{1}{2\alpha_P} d_{N-1}^{n_1 n_2} + |\mathbf{P} - \mathbf{B}|_x d_N^{n_1 n_2} + (N+1) d_{N+1}^{n_1 n_2} \end{aligned}$$

and

$$\begin{aligned} d_N^{n_1+1, n_2, n_3} &= \frac{1}{2\alpha_R} d_{N-1}^{n_1 n_2 n_3} + |\mathbf{R} - \mathbf{A}|_x d_N^{n_1 n_2 n_3} + (N+1) d_{N+1}^{n_1 n_2 n_3} \\ d_N^{n_1, n_2+1, n_3} &= \frac{1}{2\alpha_R} d_{N-1}^{n_1 n_2 n_3} + |\mathbf{R} - \mathbf{B}|_x d_N^{n_1 n_2 n_3} + (N+1) d_{N+1}^{n_1 n_2 n_3} \\ d_N^{n_1, n_2, n_3+1} &= \frac{1}{2\alpha_R} d_{N-1}^{n_1 n_2 n_3} + |\mathbf{R} - \mathbf{C}|_x d_N^{n_1 n_2 n_3} + (N+1) d_{N+1}^{n_1 n_2 n_3} \end{aligned}$$

respectively. The first elements are given by

$$\begin{aligned} d_0^0 &= 1 \\ d_0^{00} &= 1 \\ d_0^{000} &= 1 \end{aligned}$$

By using the above formalisms, it is straightforward to express the doublet of primitive GTO's as

$$[ij] = E_{ij} \sum_{N=0}^{n_1+n_2} \sum_{L=0}^{l_1+l_2} \sum_{M=0}^{m_1+m_2} d_N^{n_1 n_2} d_L^{l_1 l_2} d_M^{m_1 m_2} \Lambda_N(x_P) \Lambda_L(y_P) \Lambda_M(z_P) e^{-\alpha_P r_P^2}$$

Analogously, the triplet of primitive GTO's is given by

$$[ijk] = E_{ijk} \sum_{N=0}^{n_1+n_2+n_3} \sum_{L=0}^{l_1+l_2+l_3} \sum_{M=0}^{m_1+m_2+m_3} d_N^{n_1 n_2 n_3} d_L^{l_1 l_2 l_3} d_M^{m_1 m_2 m_3} \Lambda_N(x_R) \Lambda_L(y_R) \Lambda_M(z_R) e^{-\alpha_R r_R^2}$$

The multiplicative constants are given by

$$\begin{aligned} E_{ij}(\alpha_1, \alpha_2) &= \exp \left[ -\frac{\alpha_1 \alpha_2}{\alpha_1 + \alpha_2} |\mathbf{A} - \mathbf{B}|^2 \right] \\ E_{ijk}(\alpha_1, \alpha_2, \alpha_3) &= \exp \left[ -\frac{\alpha_1 \alpha_2}{\alpha_1 + \alpha_2} |\mathbf{A} - \mathbf{B}|^2 \right] \exp \left[ -\frac{(\alpha_1 + \alpha_2) \alpha_3}{\alpha_1 + \alpha_2 + \alpha_3} |\mathbf{P} - \mathbf{C}|^2 \right] \end{aligned}$$

### 15.4.3 One-Body Integrals over Hermite Functions

The fundamental Hermite integrals that appear during computations of any kind of one-body integrals over GTO's are as follows

$$[NLM|\Theta(1)] \equiv \int d\mathbf{r}_1 \Theta(\mathbf{r}_1) \Lambda_N(x_{1P}; \alpha_P) \Lambda_L(y_{1P}; \alpha_P) \Lambda_M(z_{1P}; \alpha_P) e^{-\alpha_P r_{1P}^2}$$

It immediately follows that the overlap, dipole, quadrupole and potential integrals are given as

$$\begin{aligned} [NLM|1] &= \delta_{N0} \delta_{L0} \delta_{M0} \left( \frac{\pi}{\alpha_P} \right)^{3/2} \\ [NLM|x_C] &= [\delta_{N1} + |\mathbf{PC}|_x \delta_{N0}] \delta_{L0} \delta_{M0} \left( \frac{\pi}{\alpha_P} \right)^{3/2} \\ [NLM|x_C^2] &= \left[ 2\delta_{N2} + 2|\mathbf{PC}|_x \delta_{N1} + \left( |\mathbf{PC}|_x^2 + \frac{1}{2\alpha_P} \right) \delta_{N0} \right] \delta_{L0} \delta_{M0} \left( \frac{\pi}{\alpha_P} \right)^{3/2} \\ [NLM|x_C y_C] &= (\delta_{N1} + |\mathbf{PC}|_x \delta_{N0}) (\delta_{L1} + |\mathbf{PC}|_y \delta_{L0}) \delta_{M0} \left( \frac{\pi}{\alpha_P} \right)^{3/2} \\ [NLM|r_C^{-1}] &= \frac{2\pi}{\alpha_P} R_{NLM} \end{aligned}$$

The coefficients  $R_{NLM}$  are discussed in separate section below.

### 15.4.4 Two-Body Integrals over Hermite Functions

The fundamental Hermite integrals that appear during computations of any kind of two-electron integrals over GTO's are as follows

$$[N_1 L_2 M_2 | N_2 L_2 M_2] \equiv \iint d\mathbf{r}_1 d\mathbf{r}_2 \Lambda_{N_1}(x_{1P}; \alpha_P) \Lambda_{L_1}(y_{1P}; \alpha_P) \Lambda_{M_1}(z_{1P}; \alpha_P) \Lambda_{N_2}(x_{2Q}; \alpha_Q) \Lambda_{L_2}(y_{2Q}; \alpha_Q) \Lambda_{M_2}(z_{2Q}; \alpha_Q) e^{-\alpha_P r_{1P}^2 - \alpha_Q r_{2Q}^2}$$

The above formula dramatically reduces to the following

$$[N_1 L_2 M_2 | N_2 L_2 M_2] = \lambda (-)^{N_2+L_2+M_2} R_{N_1+N_2, L_1+L_2, M_1+M_2}$$

with

$$\lambda \equiv \frac{2\pi^{5/2}}{\alpha_P \alpha_Q \sqrt{\alpha_P + \alpha_Q}}$$

To compute the  $R_{N_1+N_2, L_1+L_2, M_1+M_2}$  coefficients, the parameter  $T$  is given by

$$T = \frac{\alpha_P \alpha_Q}{\alpha_P + \alpha_Q} |\mathbf{P} - \mathbf{Q}|^2$$

### 15.4.5 The R(N,L,M) Coefficients

The  $R$  coefficients are defined as

$$R_{NLM} \equiv \left( \frac{\partial}{\partial a} \right)^N \left( \frac{\partial}{\partial b} \right)^L \left( \frac{\partial}{\partial c} \right)^M \int_0^1 e^{-Tu^2} du$$

with

$$T \equiv \alpha (a^2 + b^2 + c^2)$$

By extending the above definition to more general

$$R_{NLMj} \equiv (-\sqrt{\alpha})^{N+L+M} (-2\alpha)^j \int_0^1 u^{N+L+M+2j} H_N(au\sqrt{\alpha}) H_L(bu\sqrt{\alpha}) H_M(cu\sqrt{\alpha}) e^{-Tu^2} du$$

one can see that

$$R_{000j} = (-2\alpha)^j F_j(T)$$

The Boys function is here given by

$$F_j(T) \equiv \int_0^1 u^{2j} e^{-Tu^2} du$$

and its efficient implementation can be discussed elsewhere. In Psi4, `psi::Taylor_Fjt` class is used for this purpose.

Now, it is possible to show that the following recursion relationships are true:

$$\begin{aligned} R_{0,0,M+1,j} &= cR_{0,0,M,j+1} + MR_{0,0,M-1,j+1} \\ R_{0,L+1,M,j} &= bR_{0,L,M,j+1} + LR_{0,L-1,M,j+1} \\ R_{N+1,L,M,j} &= aR_{N,L,M,j+1} + NR_{N-1,L,M,j+1} \end{aligned}$$

This scheme is implemented in OEPDev.

## 15.4.6 Function Documentation

15.4.6.1 `double oepdev::d_N_n1_n2 ( int N, int n1, int n2, double PA, double PB, double aP )`

Compute McMurchie-Davidson-Hermite (MDH) coefficient for binomial expansion.

Parameters

<i>N</i>	- increment in the summation of MDH series
<i>n1</i>	- angular momentum of first function
<i>n2</i>	- angular momentum of second function
<i>PA</i>	- cartesian component of P-A distance
<i>PB</i>	- cartesian component of P-B distance
<i>aP</i>	- free parameter of MDH expansion

Returns

the McMurchie-Davidson-Hermite coefficient

15.4.6.2 `void oepdev::make_mdh_D1_coeff ( int n1, double aPd, double * buffer )`

Compute the McMurchie-Davidson-Hermite coefficients for monomial expansion.

Parameters

<i>n1</i>	- angular momentum of first function
<i>aPd</i>	- parameter equal to 0.500/Pa where Pa is exponent
<i>buffer</i>	- the McMurchie-Davidson-Hermite 3-dimensional array (raveled to vector): <ul style="list-style-type: none"> <li>axis 0: dimension 3 (x, y or z Cartesian component)</li> <li>axis 1: dimension n1+1 (0 to n1)</li> <li>axis 2: dimension n1+1 (0 to n1)</li> </ul>

See Also

[D1\\_INDEX](#)

15.4.6.3 `void oepdev::make_mdh_D2_coeff ( int n1, int n2, double aPd, double * PA, double * PB, double * buffer )`

Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion.

## Parameters

<i>n1</i>	- angular momentum of first function
<i>n2</i>	- angular momentum of second function
<i>aPd</i>	- parameter equal to 0.500/ <i>Pa</i> where <i>Pa</i> is exponent
<i>PA</i>	- cartesian components of P-A distance
<i>PB</i>	- cartesian components of P-B distance
<i>buffer</i>	- the McMurchie-Davidson-Hermite 4-dimensional array (raveled to vector): <ul style="list-style-type: none"> <li>• axis 0: dimension 3 (x, y or z Cartesian component)</li> <li>• axis 1: dimension <i>n1</i>+1 (0 to <i>n1</i>)</li> <li>• axis 2: dimension <i>n2</i>+1 (0 to <i>n2</i>)</li> <li>• axis 3: dimension <i>n1</i>+<i>n2</i>+1 (0 to <i>n1</i>+<i>n2</i>)</li> </ul>

## See Also

[D2\\_INDEX](#)

15.4.6.4 `void oepdev::make_mdh_D2_coeff_explicit_recursion ( int n1, int n2, double aP, double * PA, double * PB, double * buffer )`

Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion by explicit recursion. This function makes the same changes to buffers as [oepdev::make\\_mdh\\_D2\\_coeff](#), but implements it through explicit recursion by calls to [oepdev::d\\_N\\_n1\\_n2](#). Therefore, it is slightly slower. Here for debugging purposes.

## Parameters

<i>n1</i>	- angular momentum of first function
<i>n2</i>	- angular momentum of second function
<i>aPd</i>	- parameter equal to 0.500/ <i>Pa</i> where <i>Pa</i> is exponent
<i>PA</i>	- cartesian components of P-A distance
<i>PB</i>	- cartesian components of P-B distance
<i>buffer</i>	- the McMurchie-Davidson-Hermite 4-dimensional array (raveled to vector): <ul style="list-style-type: none"> <li>• axis 0: dimension 3 (x, y or z Cartesian component)</li> <li>• axis 1: dimension <i>n1</i>+1 (0 to <i>n1</i>)</li> <li>• axis 2: dimension <i>n2</i>+1 (0 to <i>n2</i>)</li> <li>• axis 3: dimension <i>n1</i>+<i>n2</i>+1 (0 to <i>n1</i>+<i>n2</i>)</li> </ul>

## See Also

[D2\\_INDEX](#)

15.4.6.5 `void oepdev::make_mdh_D3_coeff ( int n1, int n2, int n3, double aPd, double * PA, double * PB, double * PC, double * buffer )`

Compute the McMurchie-Davidson-Hermite coefficients for trinomial expansion.

## Parameters

<i>n1</i>	- angular momentum of first function
<i>n2</i>	- angular momentum of second function
<i>n3</i>	- angular momentum of third function
<i>aPd</i>	- parameter equal to 0.500/ <i>Pa</i> where <i>Pa</i> is exponent
<i>PA</i>	- cartesian components of P-A distance
<i>PB</i>	- cartesian components of P-B distance
<i>PC</i>	- cartesian components of P-C distance
<i>buffer</i>	- the McMurchie-Davidson-Hermite 5-dimensional array (raveled to vector): <ul style="list-style-type: none"> <li>• axis 0: dimension 3 (x, y or z Cartesian component)</li> <li>• axis 1: dimension <i>n1</i>+1 (0 to <i>n1</i>)</li> <li>• axis 2: dimension <i>n2</i>+1 (0 to <i>n2</i>)</li> <li>• axis 3: dimension <i>n3</i>+1 (0 to <i>n3</i>)</li> <li>• axis 4: dimension <i>n1</i>+<i>n2</i>+<i>n3</i>+1 (0 to <i>n1</i>+<i>n2</i>+<i>n3</i>)</li> </ul>

## See Also

[D3\\_INDEX](#)

15.4.6.6 void oepdev::make\_mdh\_R\_coeff ( int *N*, int *L*, int *M*, double *alpha*, double *a*, double *b*, double *c*, double \* *F*, double \* *buffer* )

Compute the McMurchie-Davidson R coefficients.

## Parameters

<i>N</i>	- increment in the summation of MDH series along x direction
<i>L</i>	- increment in the summation of MDH series along y direction
<i>M</i>	- increment in the summation of MDH series along z direction
<i>alpha</i>	- alpha parameter of R coefficient
<i>a</i>	- x component of PQ vector of R coefficient
<i>b</i>	- y component of PQ vector of R coefficient
<i>c</i>	- z component of PQ vector of R coefficient
<i>F</i>	- array of Boys function values for given alpha and PQ
<i>buffer</i>	- the McMurchie-Davidson 4-dimensional array (raveled to vector): <ul style="list-style-type: none"> <li>• axis 0: dimension <i>N</i>+1</li> <li>• axis 1: dimension <i>L</i>+1</li> <li>• axis 2: dimension <i>M</i>+1</li> <li>• axis 3: dimension <i>N</i>+<i>L</i>+<i>M</i>+1 (<i>j</i>-th element)</li> </ul>

## 15.5 The Three-Dimensional Vector Fields Library

Handles all sorts of scalar distributions in 3D Euclidean space, such as general vector potentials defined at particular collection of points. In this Module, you will also find handling both random and ordered points collections in a form of a G09 cube, as well as handling G09 Cube files. You will also find solvers used to fit the generalized multipole moments of a generalized density distribution, such as the electrostatic potential (ESP) fitting method. Located at `oepdev/lib3d`.

### Classes

- class `oepdev::MultipoleConvergence`  
*Multipole Convergence.*
- class `oepdev::DMTPole`  
*Distributed Multipole Analysis Container and Computer. Abstract Base.*
- class `oepdev::CMM`  
*Cumulative Atomic Multipole Moments.*
- class `oepdev::ESPSolver`  
*Charges from Electrostatic Potential (ESP). A solver-type class.*
- class `oepdev::Points3DIterator`  
*Iterator over a collection of points in 3D space. Abstract base.*
- class `oepdev::CubePoints3DIterator`  
*Iterator over a collection of points in 3D space. g09 Cube-like order.*
- class `oepdev::RandomPoints3DIterator`  
*Iterator over a collection of points in 3D space. Random collection.*
- class `oepdev::PointsCollection3D`  
*Collection of points in 3D space. Abstract base.*
- class `oepdev::RandomPointsCollection3D`  
*Collection of random points in 3D space.*
- class `oepdev::CubePointsCollection3D`  
*G09 cube-like ordered collection of points in 3D space.*
- class `oepdev::Field3D`  
*General Vector Field in 3D Space. Abstract base.*
- class `oepdev::ElectrostaticPotential3D`  
*Electrostatic potential of a molecule.*
- class `oepdev::OEPotential3D< T >`  
*Class template for OEP 3D fields.*

### Typedefs

- using `oepdev::SharedField3D` = `std::shared_ptr< oepdev::Field3D >`

### Functions

- `oepdev::OEPotential3D< T >::OEPotential3D` (const int &ndim, const int &np, const double &padding, std::shared\_ptr< T > oep, const std::string &oepType)  
*Construct random spherical collection of 3D field of type T.*
- `oepdev::OEPotential3D< T >::OEPotential3D` (const int &ndim, 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 3D field of type T.*
- virtual `oepdev::OEPotential3D< T >::~~OEPotential3D` ()

*Destructor.*

- virtual void `oepdev::OEPotential3D< T >::print ()` const

*Print information of the object to Psi4 output.*

- virtual `std::shared_ptr< psi::Vector > oepdev::OEPotential3D< T >::compute_xyz` (const double &x, const double &y, const double &z)

*Compute a value of 3D field at point (x, y, z)*

### 15.5.1 Detailed Description

Handles all sorts of scalar distributions in 3D Euclidean space, such as general vector potentials defined at particular collection of points. In this Module, you will also find handling both random and ordered points collections in a form of a G09 cube, as well as handling G09 Cube files. You will also find solvers used to fit the generalized multipole moments of a generalized density distribution, such as the electrostatic potential (ESP) fitting method. Located at `oepdev/lib3d`.

### 15.5.2 Function Documentation

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

Construct random spherical collection of 3D field of type T.

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

**Parameters**

<i>ndim</i>	- dimensionality of 3D field (1: scalar field, >2: vector field)
<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

**15.5.2.2** `template<class T > oepdev::OEPotential3D< T >::OEPotential3D ( const int & ndim, 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 3D field of type T.

The points are generated according to Gaussian cube file format.

**Parameters**

<i>ndim</i>	- dimensionality of 3D field (1: scalar field, >2: vector 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>oep</i>	- OEP object of type T



<i>oepType</i>	- type of OEP
<i>options</i>	- Psi4 options object

## 15.6 The Density Functional Theory Library

Implements the OEPDev ab initio DFT methods. Located at `oepdev/libdft`. Currently, this library is empty.

Implements the OEPDev ab initio DFT methods. Located at `oepdev/libdft`. Currently, this library is empty.

## 15.7 The OEPDev Utilities

Contains utility functions such as printing OEPDev preamble to the output file, class for wavefunction union, DIIS converger, CPHF Solver, SCF solver for external electrostatic perturbations, and others. You will also find here various iterators to go through orbital shells while computing ERI, or iterators over ERI itself. Located at `oepdev/libutil`.

### Classes

- class `oepdev::CPHF`  
*CPHF solver class.*
- class `oepdev::DIISManager`  
*DIIS manager.*
- class `oepdev::ShellCombinationsIterator`  
*Iterator for Shell Combinations. Abstract Base.*
- class `oepdev::AOIntegralsIterator`  
*Iterator for AO Integrals. Abstract Base.*
- class `oepdev::AllAOShellCombinationsIterator_4`  
*Loop over all possible ERI shells in a shell quartet.*
- class `oepdev::AllAOShellCombinationsIterator_2`  
*Loop over all possible ERI shells in a shell doublet.*
- class `oepdev::AllAOIntegralsIterator_4`  
*Loop over all possible ERI within a particular shell quartet.*
- class `oepdev::AllAOIntegralsIterator_2`  
*Loop over all possible ERI within a particular shell doublet.*
- struct `oepdev::PerturbCharges`  
*Structure to hold perturbing charges.*
- class `oepdev::RHFPerturbed`  
*RHF theory under electrostatic perturbation.*
- struct `oepdev::ABCD`  
*Simple structure to hold the Fourier series expansion coefficients.*
- struct `oepdev::Fourier9`  
*Simple structure to hold the Fourier series expansion coefficients for N=4.*
- class `oepdev::UnitaryOptimizer`  
*Find the optimum unitary matrix of quadratic matrix equation.*
- class `oepdev::UnitaryOptimizer_4_2`  
*Find the optimum unitary matrix for quartic-quadratic matrix equation with trace.*
- class `oepdev::WavefunctionUnion`  
*Union of two Wavefunction objects.*

### Macros

- `#define OEPDEV_USE_PSI4_DIIS_MANAGER 0`  
*Use DIIS from Psi4 (1) or OEPDev (0)?*
- `#define OEPDEV_MAX_AM 8`  
*L\_max.*
- `#define OEPDEV_N_MAX_AM 17`  
*2L\_max+1*
- `#define OEPDEV_CRIT_ERI 1e-9`  
*ERI criterion for E12, E34, E123 and lambda\*EXY coefficients.*
- `#define OEPDEV_SIZE_BUFFER_R 250563`

- Size of R buffer ( $OEPDEV\_N\_MAX\_AM * OEPDEV\_N\_MAX\_AM * OEPDEV\_N\_MAX\_AM * OEPDEV\_N\_MAX\_AM * 3$ )
- #define `OEPDEV_SIZE_BUFFER_D2` 3264
- Size of D2 buffer ( $3 * (OEPDEV\_MAX\_AM + 1) * (OEPDEV\_MAX\_AM + 1) * OEPDEV\_N\_MAX\_AM$ )

## Typedefs

- using `oepdev::SharedShellsIterator` = `std::shared_ptr< ShellCombinationsIterator >`  
Iterator over shells as shared pointer.
- using `oepdev::SharedAOIntsIterator` = `std::shared_ptr< AOIntegralsIterator >`  
Iterator over AO integrals as shared pointer.

## Functions

- void `oepdev::preamble` (void)  
Print preamble for module OEPDEV.
- template<typename... Args>  
`std::string oepdev::string_sprintf` (const char \*format, Args...args)  
Format string output. Example: `std::string text = oepdev::string_sprintf("Test %3d, %13.5f", 5, -10.5425);`.
- `std::shared_ptr< SuperFunctional > oepdev::create_superfunctional` (std::string name, Options &options)  
Set up DFT functional.
- `std::shared_ptr< Molecule > oepdev::extract_monomer` (std::shared\_ptr< const Molecule > molecule, int id)  
Extract molecule from dimer.
- `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.
- double `oepdev::average_moment` (std::shared\_ptr< psi::Vector > moment)  
Compute the scalar magnitude of multipole moment.

### 15.7.1 Detailed Description

Contains utility functions such as printing OEPDev preamble to the output file, class for wavefunction union, D-IIS converger, [CPHF](#) Solver, SCF solver for external electrostatic perturbations, and others. You will also find here various iterators to go through orbital shells while computing ERI, or iterators over ERI itself. Located at `oepdev/libutil`.

### 15.7.2 Function Documentation

#### 15.7.2.1 double oepdev::average\_moment ( std::shared\_ptr< psi::Vector > moment )

Compute the scalar magnitude of multipole moment.

##### Parameters

<i>moment</i>	- multipole moment vector with unique matrix elements. Now supported only for dipole and quadrupole.
---------------	--

## Returns

- the average multipole moment value.

The magnitudes of multipole moments are defined here as follows:

- The dipole moment magnitude is just a norm

$$|\mu| \equiv \sqrt{\mu_x^2 + \mu_y^2 + \mu_z^2}$$

- The quadrupole moment magnitude refers to the traceless moment in Buckingham convention

$$|\Theta| \equiv \sqrt{\Theta_{zz}^2 + \frac{1}{3}(\Theta_{xx} - \Theta_{yy})^2 + \frac{4}{3}(\Theta_{xy}^2 + \Theta_{xz}^2 + \Theta_{yz}^2)}$$

In the above equation, the quadrupole moment elements refer to its traceless form.

### 15.7.2.2 `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.

## Examples:

[example\\_scf\\_perturb.cc](#).

### 15.7.2.3 `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

### 15.7.2.4 `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

## 15.8 The OEPDev Testing Platform Library

Testing platform at C++ level of code. You should add more tests here when developing new functionalities, theories or models. Located at `oepdev/libtest`.

### Classes

- class `oepdev::test::Test`  
*Manages test routines.*

### 15.8.1 Detailed Description

Testing platform at C++ level of code. You should add more tests here when developing new functionalities, theories or models. Located at `oepdev/libtest`.





## Chapter 16

# Namespace Documentation

### 16.1 oepdev Namespace Reference

OEPDev module namespace.

#### Classes

- class [MultipoleConvergence](#)  
*Multipole Convergence.*
- class [DMTPole](#)  
*Distributed Multipole Analysis Container and Computer. Abstract Base.*
- class [CAMM](#)  
*Cumulative Atomic Multipole Moments.*
- class [ESPSolver](#)  
*Charges from Electrostatic Potential (ESP). A solver-type class.*
- 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 [Field3D](#)  
*General Vector Field in 3D Space. Abstract base.*
- class [ElectrostaticPotential3D](#)  
*Electrostatic potential of a molecule.*
- class [OEPotential3D](#)  
*Class template for OEP 3D fields.*
- class [GenEffPar](#)  
*Generalized Effective Fragment Parameters. Container Class.*
- class [GenEffFrag](#)  
*Generalized Effective Fragment. Container Class.*

- class [GenEffParFactory](#)  
*Generalized Effective Fragment Factory. Abstract Base.*
- class [PolarGEFactory](#)  
*Polarization GEFP Factory. Abstract Base.*
- class [AbInitioPolarGEFactory](#)  
*Polarization GEFP Factory from First Principles. Hartree-Fock Approximation.*
- class [FFAbInitioPolarGEFactory](#)  
*Polarization GEFP Factory from First Principles: Finite-Difference Model. Arbitrary level of theory.*
- class [GeneralizedPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [UniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [NonUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [LinearUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [QuadraticUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [LinearNonUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [QuadraticNonUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [LinearGradientNonUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [QuadraticGradientNonUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [UnitaryTransformedMOPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Scaling of MO Space.*
- class [TwoElectronInt](#)  
*General Two Electron Integral.*
- class [ERI\\_1\\_1](#)  
*2-centre ERI of the form  $\langle a|O(2)|b \rangle$  where  $O(2) = 1/r^{12}$ .*
- class [ERI\\_2\\_2](#)  
*4-centre ERI of the form  $\langle ab|O(2)|cd \rangle$  where  $O(2) = 1/r^{12}$ .*
- class [ERI\\_3\\_1](#)  
*4-centre ERI of the form  $\langle abc|O(2)|d \rangle$  where  $O(2) = 1/r^{12}$ .*
- struct [OEPTyp](#)  
*Container to handle the type of One-Electron Potentials.*
- class [OEPotential](#)  
*Generalized One-Electron Potential: Abstract base.*
- class [ElectrostaticEnergyOEPotential](#)  
*Generalized One-Electron Potential for Electrostatic Energy.*
- class [RepulsionEnergyOEPotential](#)  
*Generalized One-Electron Potential for Pauli Repulsion Energy.*
- class [ChargeTransferEnergyOEPotential](#)  
*Generalized One-Electron Potential for Charge-Transfer Interaction Energy.*
- class [EETCouplingOEPotential](#)  
*Generalized One-Electron Potential for EET coupling calculations.*
- class [GeneralizedDensityFit](#)  
*Generalized Density Fitting Scheme. Abstract Base.*
- class [SingleGeneralizedDensityFit](#)

- Generalized Density Fitting Scheme - Single Fit.*
- class [DoubleGeneralizedDensityFit](#)
  - Generalized Density Fitting Scheme - Double Fit.*
- class [TwoBodyAOInt](#)
- class [IntegralFactory](#)
  - Extended [IntegralFactory](#) for computing integrals.*
- class [PotentialInt](#)
  - Computes potential integrals.*
- 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 [ChargeTransferEnergySolver](#)
  - Compute the Charge-Transfer interaction energy between unperturbed wavefunctions.*
- class [CPHF](#)
  - [CPHF](#) solver class.*
- class [DIISManager](#)
  - DIIS manager.*
- class [ShellCombinationsIterator](#)
  - Iterator for Shell Combinations. Abstract Base.*
- class [AOIntegralsIterator](#)
  - Iterator for AO Integrals. Abstract Base.*
- class [AllAOShellCombinationsIterator\\_4](#)
  - Loop over all possible ERI shells in a shell quartet.*
- class [AllAOShellCombinationsIterator\\_2](#)
  - Loop over all possible ERI shells in a shell doublet.*
- class [AllAOIntegralsIterator\\_4](#)
  - Loop over all possible ERI within a particular shell quartet.*
- class [AllAOIntegralsIterator\\_2](#)
  - Loop over all possible ERI within a particular shell doublet.*
- struct [PerturbCharges](#)
  - Structure to hold perturbing charges.*
- class [RHPerturbed](#)
  - RHF theory under electrostatic perturbation.*
- struct [ABCD](#)
  - Simple structure to hold the Fourier series expansion coefficients.*
- struct [Fourier9](#)
  - Simple structure to hold the Fourier series expansion coefficients for N=4.*
- class [UnitaryOptimizer](#)
  - Find the optimum unitary matrix of quadratic matrix equation.*
- class [UnitaryOptimizer\\_4\\_2](#)
  - Find the optimum unitary matrix for quartic-quadratic matrix equation with trace.*
- class [WavefunctionUnion](#)
  - Union of two Wavefunction objects.*

## Typedefs

- using **SharedField3D** = std::shared\_ptr< [oepdev::Field3D](#) >
- using **SharedWavefunction** = std::shared\_ptr< Wavefunction >
- using **SharedBasisSet** = std::shared\_ptr< BasisSet >
- using **SharedMatrix** = std::shared\_ptr< Matrix >
- using **SharedVector** = std::shared\_ptr< Vector >
- using **SharedDMTPole** = std::shared\_ptr< [DMTPole](#) >
- using **SharedWavefunctionUnion** = std::shared\_ptr< [WavefunctionUnion](#) >
- using **SharedOEPotential** = std::shared\_ptr< [OEPotential](#) >
- using **SharedIntegralFactory** = std::shared\_ptr< [IntegralFactory](#) >
- using **SharedTwoBodyAOInt** = std::shared\_ptr< [TwoBodyAOInt](#) >
- using [SharedShellsIterator](#) = std::shared\_ptr< [ShellCombinationsIterator](#) >  
*Iterator over shells as shared pointer.*
- using [SharedAOIntsIterator](#) = std::shared\_ptr< [AOIntegralsIterator](#) >  
*Iterator over AO integrals as shared pointer.*
- 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

- **dipoles\_** ({} )
- **quadrupoles\_** ({} )
- **octupoles\_** ({} )
- **hexadecapoles\_** ({} )
- **mplnts\_** ({} )
- **n\_max\_am\_** (2 \*max\_am\_+1)
- double **d\_N\_n1\_n2** (int N, int n1, int n2, double PA, double PB, double aP)  
*Compute McMurchie-Davidson-Hermite (MDH) coefficient for binomial expansion.*
- void [make\\_mdh\\_D2\\_coeff\\_explicit\\_recursion](#) (int n1, int n2, double aP, double \*PA, double \*PB, double \*buffer)  
*Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion by explicit recursion. This function makes the same changes to buffers as [oepdev::make\\_mdh\\_D2\\_coeff](#), but implements it through explicit recursion by calls to [oepdev::d\\_N\\_n1\\_n2](#). Therefore, it is slightly slower. Here for debugging purposes.*
- void [make\\_mdh\\_D1\\_coeff](#) (int n1, double aPd, double \*buffer)  
*Compute the McMurchie-Davidson-Hermite coefficients for monomial expansion.*
- void [make\\_mdh\\_D2\\_coeff](#) (int n1, int n2, double aPd, double \*PA, double \*PB, double \*buffer)  
*Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion.*
- void [make\\_mdh\\_D3\\_coeff](#) (int n1, int n2, int n3, double aPd, double \*PA, double \*PB, double \*PC, double \*buffer)  
*Compute the McMurchie-Davidson-Hermite coefficients for trinomial expansion.*
- void [make\\_mdh\\_R\\_coeff](#) (int N, int L, int M, double alpha, double a, double b, double c, double \*F, double \*buffer)  
*Compute the McMurchie-Davidson R coefficients.*
- constexpr std::complex< double > **operator""\_i** (unsigned long long d)
- constexpr std::complex< double > **operator""\_i** (long double d)
- 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.*

- `double average_moment (std::shared_ptr< psi::Vector > moment)`

*Compute the scalar magnitude of multipole moment.*

- `template<typename... Args>  
std::string string_sprintf (const char *format, Args...args)`

*Format string output. Example: `std::string text = oepdev::string_sprintf("Test %3d, %13.5f", 5, -10.5425);`*

### 16.1.1 Detailed Description

OEPDev module namespace. Contains all the functionalities for the development of the Generalized One-Electrode Potentials (OEP's).

## 16.2 psi Namespace Reference

Psi4 package namespace.

### Typedefs

- 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.*

### 16.2.1 Detailed Description

Psi4 package namespace. Contains all Psi4 functionalities.

### 16.2.2 Function Documentation

#### 16.2.2.1 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 repulsion and CT 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). The list of implemented models can be found in [Implemented Models](#) .

#### Parameters

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

#### Returns

psi::SharedWavefunction (either ref\_wfn or wavefunction union)

#### 16.2.2.2 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 17

# Class Documentation

### 17.1 oepdev::ABCD Struct Reference

Simple structure to hold the Fourier series expansion coefficients.

```
#include <unitary_optimizer.h>
```

#### Public Attributes

- double **A**
- double **B**
- double **C**
- double **D**

#### 17.1.1 Detailed Description

Simple structure to hold the Fourier series expansion coefficients.

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

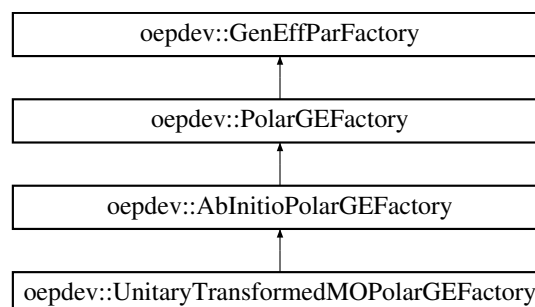
- oepdev/libutil/[unitary\\_optimizer.h](#)

### 17.2 oepdev::AbInitioPolarGEFactory Class Reference

Polarization GEFP Factory from First Principles. Hartree-Fock Approximation.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::AbInitioPolarGEFactory:



## Public Member Functions

- **AbInitioPolarGEFactory** (std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- virtual std::shared\_ptr  
< [GenEffPar](#) > [compute](#) (void)  
*Compute the density matrix susceptibility tensors.*

## Additional Inherited Members

### 17.2.1 Detailed Description

Polarization GEFP Factory from First Principles. Hartree-Fock Approximation.

Implements creation of the density matrix susceptibility tensors for which  $\mathbf{X} = \mathbf{1}$ . Guarantees the idempotency of the density matrix up to first-order in LCAO-MO variation. The density matrix susceptibility tensor is represented by:

$$\delta D_{\alpha\beta} = \sum_i \mathbf{B}_{\alpha\beta}^{(i;1)} \cdot \mathbf{F}(\mathbf{r}_i)$$

where  $\mathbf{B}_{\alpha\beta}^{(i;1)}$  is the density matrix dipole polarizability defined for the distributed LMO site at  $\mathbf{r}_i$ . Its explicit form is given by

$$\mathbf{B}_{\alpha\beta}^{(i;1)} = C_{\alpha i}^{(0)} \mathbf{b}_{\beta}^{(i;1)} C_{\beta i}^{(0)} \mathbf{b}_{\alpha}^{(i;1)} - \sum_{\gamma} \left( D_{\alpha\gamma}^{(0)} C_{\beta i}^{(0)} + D_{\beta\gamma}^{(0)} C_{\alpha i}^{(0)} \right) \mathbf{b}_{\gamma}^{(i;1)}$$

where the susceptibility of the LCAO-MO coefficient is given by

$$b_{\alpha;w}^{(i;1)} = \frac{1}{4} \sum_u^{x,y,z} [\alpha_i]_{uw} \left[ [\mathbf{L}_i]_{\text{Left}}^{-1} \right]_{u;\alpha}$$

for  $w = x, y, z$ . The auxiliary tensor  $\mathbb{L}$  is defined as

$$\mathbb{L} = \mathbf{C}^{(0)\text{T}} \cdot \mathbb{M} \cdot (\mathbf{1} - \mathbf{D}^{(0)})$$

where  $\mathbb{M}$  is the dipole integral vector of matrices in AO representation. The left inverse of the  $i$ -th element is defined as

$$[\mathbf{L}_i]_{\text{Left}}^{-1} \equiv [\mathbf{L}_i^{\text{T}} \cdot \mathbf{L}_i]^{-1} \cdot \mathbf{L}_i^{\text{T}}$$

Note that  $\mathbf{L}_i \equiv [\mathbb{L}]_i$  is a  $n \times 3$  matrix, whereas its left inverse is a  $3 \times n$  matrix with  $n$  being the size of the AO basis set.

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

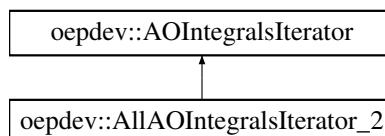
- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp\_polar\_abinitio.cc

## 17.3 oepdev::AllAOIntegralsIterator\_2 Class Reference

Loop over all possible ERI within a particular shell doublet.

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

Inheritance diagram for oepdev::AllAOIntegralsIterator\_2:





## Public Member Functions

- [AllAOIntegralsIterator\\_2](#) (const [ShellCombinationsIterator](#) \*shellIter)  
*Construct by shell iterator (const object)*
- [AllAOIntegralsIterator\\_2](#) (std::shared\_ptr< [ShellCombinationsIterator](#) > shellIter)  
*Construct by shell iterator (pointed by shared pointer)*
- void [first](#) ()  
*First iteration.*
- void [next](#) ()  
*Next iteration.*
- int [i](#) () const  
*Grab the current integral i index.*
- int [j](#) () const  
*Grab the current integral j index.*
- int [index](#) () const

## Additional Inherited Members

### 17.3.1 Detailed Description

Loop over all possible ERI within a particular shell doublet.

Constructed by providing a const reference or shared pointer to an AllAOShellCombinationsIterator object.

See Also

[AllAOShellCombinationsIterator\\_2](#)

### 17.3.2 Constructor & Destructor Documentation

#### 17.3.2.1 AllAOIntegralsIterator\_2::AllAOIntegralsIterator\_2 ( const [ShellCombinationsIterator](#) \* *shellIter* )

Construct by shell iterator (const object)

Parameters

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

#### 17.3.2.2 AllAOIntegralsIterator\_2::AllAOIntegralsIterator\_2 ( std::shared\_ptr< [ShellCombinationsIterator](#) > *shellIter* )

Construct by shell iterator (pointed by shared pointer)

Parameters

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

### 17.3.3 Member Function Documentation

#### 17.3.3.1 int oepdev::AllAOIntegralsIterator\_2::index ( void ) const [inline], [virtual]

Grab the current index of integral value stored in the buffer

Implements [oepdev::AOIntegralsIterator](#).

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

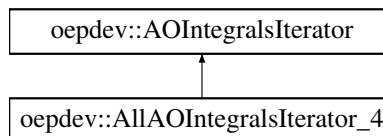
- [oepdev/libutil/integrals\\_iter.h](#)
- [oepdev/libutil/integrals\\_iter.cc](#)

## 17.4 oepdev::AllAOIntegralsIterator\_4 Class Reference

Loop over all possible ERI within a particular shell quartet.

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

Inheritance diagram for oepdev::AllAOIntegralsIterator\_4:



### Public Member Functions

- [AllAOIntegralsIterator\\_4](#) (const [ShellCombinationsIterator](#) \*shellIter)  
*Construct by shell iterator (const object)*
- [AllAOIntegralsIterator\\_4](#) (std::shared\_ptr< [ShellCombinationsIterator](#) > shellIter)  
*Construct by shell iterator (pointed by shared pointer)*
- void [first](#) ()  
*First iteration.*
- void [next](#) ()  
*Next iteration.*
- 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

### Additional Inherited Members

#### 17.4.1 Detailed Description

Loop over all possible ERI within a particular shell quartet.

Constructed by providing a const reference or shared pointer to an AllAOShellCombinationsIterator object.

See Also

[AllAOShellCombinationsIterator\\_4](#)

#### 17.4.2 Constructor & Destructor Documentation

##### 17.4.2.1 AllAOIntegralsIterator\_4::AllAOIntegralsIterator\_4 ( const ShellCombinationsIterator \* shellIter )

Construct by shell iterator (const object)

## Parameters

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

## 17.4.2.2 AllAOShellCombinationsIterator\_2::AllAOShellCombinationsIterator\_2 ( std::shared\_ptr&lt; ShellCombinationsIterator &gt; shellIter )

Construct by shell iterator (pointed by shared pointer)

## Parameters

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

## 17.4.3 Member Function Documentation

## 17.4.3.1 int oepdev::AllAOShellCombinationsIterator\_2::index ( void ) const [inline], [virtual]

Grab the current index of integral value stored in the buffer

Implements [oepdev::AOShellCombinationsIterator](#).

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

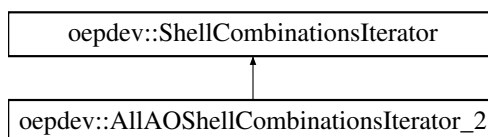
- oepdev/libutil/[integrals\\_iter.h](#)
- oepdev/libutil/[integrals\\_iter.cc](#)

## 17.5 oepdev::AllAOShellCombinationsIterator\_2 Class Reference

Loop over all possible ERI shells in a shell doublet.

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

Inheritance diagram for oepdev::AllAOShellCombinationsIterator\_2:



## Public Member Functions

- [AllAOShellCombinationsIterator\\_2](#) (SharedBasisSet [bs\\_1](#), SharedBasisSet [bs\\_2](#))  
*Iterate over shell doublets. Construct by providing basis sets for each axis. The basis sets must be defined for the same molecule.*
- [AllAOShellCombinationsIterator\\_2](#) (std::shared\_ptr< [IntegralFactory](#) > integrals)  
*Construct by providing integral factory.*
- [AllAOShellCombinationsIterator\\_2](#) (const [IntegralFactory](#) &integrals)
- [AllAOShellCombinationsIterator\\_2](#) (std::shared\_ptr< psi::IntegralFactory > integrals)  
*Construct by providing integral factory.*
- [AllAOShellCombinationsIterator\\_2](#) (const psi::IntegralFactory &integrals)
- void [first](#) ()  
*First iteration.*
- void [next](#) ()  
*Next iteration.*

- void `compute_shell` (std::shared\_ptr< [oepdev::TwoBodyAOInt](#) > tei) const  
*Compute ERI's for the current shell. The eris are stored in the buffer of the argument object.*
- void `compute_shell` (std::shared\_ptr< [psi::TwoBodyAOInt](#) > tei) const
- int `P` () const  
*Grab the current shell P index.*
- int `Q` () const  
*Grab the current shell Q index.*

## Additional Inherited Members

### 17.5.1 Detailed Description

Loop over all possible ERI shells in a shell doublet.

Constructed by providing [IntegralFactory](#) object or shared pointers to two basis set spaces.

### 17.5.2 Constructor & Destructor Documentation

#### 17.5.2.1 `AllAOShellCombinationsIterator_2::AllAOShellCombinationsIterator_2 ( SharedBasisSet bs_1, SharedBasisSet bs_2 )`

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

Parameters

<code>bs_1</code>	- basis set of axis 1
<code>bs_2</code>	- basis set of axis 2

#### 17.5.2.2 `oepdev::AllAOShellCombinationsIterator_2::AllAOShellCombinationsIterator_2 ( std::shared_ptr< IntegralFactory > integrals )`

Construct by providing integral factory.

Parameters

<code>integrals</code>	- OepDev integral factory object
------------------------	----------------------------------

#### 17.5.2.3 `AllAOShellCombinationsIterator_2::AllAOShellCombinationsIterator_2 ( const IntegralFactory & integrals )`

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

#### 17.5.2.4 `AllAOShellCombinationsIterator_2::AllAOShellCombinationsIterator_2 ( std::shared_ptr< psi::IntegralFactory > integrals )`

Construct by providing integral factory.

Parameters

<code>integrals</code>	- Psi4 integral factory object
------------------------	--------------------------------

## 17.5.2.5 AllAOShellCombinationsIterator\_2::AllAOShellCombinationsIterator\_2 ( const psi::IntegralFactory &amp; integrals )

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

## 17.5.3 Member Function Documentation

## 17.5.3.1 void AllAOShellCombinationsIterator\_2::compute\_shell ( std::shared\_ptr&lt; oepdev::TwoBodyAOInt &gt; tei ) const [virtual]

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
------------	----------------------------

Implements [oepdev::ShellCombinationsIterator](#).

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

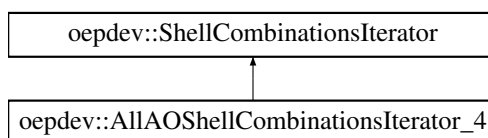
- [oepdev/libutil/integrals\\_iter.h](#)
- [oepdev/libutil/integrals\\_iter.cc](#)

## 17.6 oepdev::AllAOShellCombinationsIterator\_4 Class Reference

Loop over all possible ERI shells in a shell quartet.

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

Inheritance diagram for oepdev::AllAOShellCombinationsIterator\_4:



## Public Member Functions

- [AllAOShellCombinationsIterator\\_4](#) (SharedBasisSet [bs\\_1](#), SharedBasisSet [bs\\_2](#), SharedBasisSet [bs\\_3](#), SharedBasisSet [bs\\_4](#))  
*Iterate over shell quartets. Construct by providing basis sets for each axis. The basis sets must be defined for the same molecule.*
- [AllAOShellCombinationsIterator\\_4](#) (std::shared\_ptr< [IntegralFactory](#) > integrals)  
*Construct by providing integral factory.*
- [AllAOShellCombinationsIterator\\_4](#) (const [IntegralFactory](#) &integrals)
- [AllAOShellCombinationsIterator\\_4](#) (std::shared\_ptr< psi::IntegralFactory > integrals)  
*Construct by providing integral factory.*
- [AllAOShellCombinationsIterator\\_4](#) (const psi::IntegralFactory &integrals)
- void [first](#) ()  
*Do the first iteration.*
- void [next](#) ()  
*Do the next iteration.*
- void [compute\\_shell](#) (std::shared\_ptr< oepdev::TwoBodyAOInt > tei) const
- void [compute\\_shell](#) (std::shared\_ptr< psi::TwoBodyAOInt > tei) const

- 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.*

## Additional Inherited Members

### 17.6.1 Detailed Description

Loop over all possible ERI shells in a shell quartet.

Constructed by providing [IntegralFactory](#) object or shared pointers to four basis set spaces.

### 17.6.2 Constructor & Destructor Documentation

#### 17.6.2.1 AllAOShellCombinationsIterator\_4::AllAOShellCombinationsIterator\_4 ( SharedBasisSet *bs\_1*, SharedBasisSet *bs\_2*, SharedBasisSet *bs\_3*, SharedBasisSet *bs\_4* )

Iterate over shell quartets. 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

#### 17.6.2.2 oepdev::AllAOShellCombinationsIterator\_4::AllAOShellCombinationsIterator\_4 ( std::shared\_ptr< IntegralFactory > *integrals* )

Construct by providing integral factory.

##### Parameters

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

#### 17.6.2.3 AllAOShellCombinationsIterator\_4::AllAOShellCombinationsIterator\_4 ( const IntegralFactory & *integrals* )

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

#### 17.6.2.4 AllAOShellCombinationsIterator\_4::AllAOShellCombinationsIterator\_4 ( std::shared\_ptr< psi::IntegralFactory > *integrals* )

Construct by providing integral factory.

## Parameters

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

17.6.2.5 AllAOShellCombinationsIterator\_4::AllAOShellCombinationsIterator\_4 ( const psi::IntegralFactory & *integrals* )

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

## 17.6.3 Member Function Documentation

17.6.3.1 void AllAOShellCombinationsIterator\_4::compute\_shell ( std::shared\_ptr< oepdev::TwoBodyAOInt > *tei* ) const [virtual]

Compute integrals in a current shell. Works both for [oepdev::TwoBodyAOInt](#) and [psi::TwoBodyAOInt](#)

## Parameters

<i>tei</i>	- two body integral object
------------	----------------------------

Implements [oepdev::ShellCombinationsIterator](#).

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

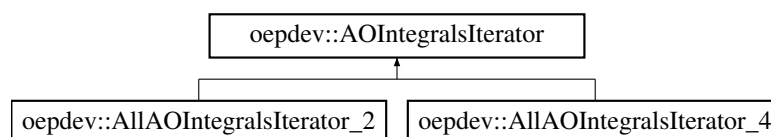
- [oepdev/libutil/integrals\\_iter.h](#)
- [oepdev/libutil/integrals\\_iter.cc](#)

## 17.7 oepdev::AOIntegralsIterator Class Reference

Iterator for AO Integrals. Abstract Base.

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

Inheritance diagram for oepdev::AOIntegralsIterator:



## Public Member Functions

- [AOIntegralsIterator](#) ()  
*Base Constructor.*
- virtual [~AOIntegralsIterator](#) ()  
*Base Destructor.*
- virtual void [first](#) (void)=0  
*Do the first iteration.*
- virtual void [next](#) (void)=0  
*Do the next iteration.*
- virtual int [i](#) (void) const  
*Grab i-th index.*
- virtual int [j](#) (void) const

- *Grab j-th index.*
- virtual int [k](#) (void) const
- *Grab k-th index.*
- virtual int [l](#) (void) const
- *Grab l-th index.*
- virtual int [index](#) (void) const =0
- *Grab index in the integral buffer.*
- virtual bool [is\\_done](#) (void)
- *Returns the status of an iterator.*

## Static Public Member Functions

- static std::shared\_ptr  
< [AOIntegralsIterator](#) > [build](#) (const [ShellCombinationsIterator](#) \*shellIter, std::string mode="ALL")
- static std::shared\_ptr  
< [AOIntegralsIterator](#) > [build](#) (std::shared\_ptr< [ShellCombinationsIterator](#) > shellIter, std::string mode="ALL")

## Protected Attributes

- bool [done](#)
- *The status of an iterator.*

### 17.7.1 Detailed Description

Iterator for AO Integrals. Abstract Base.

### 17.7.2 Member Function Documentation

**17.7.2.1** `std::shared_ptr< AOIntegralsIterator > AOIntegralsIterator::build ( const ShellCombinationsIterator *  
shellIter, std::string mode = "ALL" ) [static]`

Build AO integrals iterator from current state of iterator over shells

#### Parameters

<i>shellIter</i>	- iterator over shells - either "ALL" or "UNIQUE" (iterate over all or unique integrals)
------------------	--

#### Returns

iterator over AO integrals

**17.7.2.2** `std::shared_ptr< AOIntegralsIterator > AOIntegralsIterator::build ( std::shared_ptr<  
ShellCombinationsIterator > shellIter, std::string mode = "ALL" ) [static]`

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

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

- [oepdev/libutil/integrals\\_iter.h](#)
- [oepdev/libutil/integrals\\_iter.cc](#)

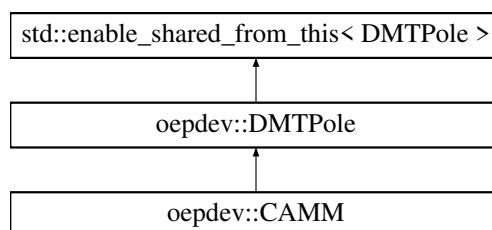


## 17.8 oepdev::CAMM Class Reference

Cumulative Atomic Multipole Moments.

```
#include <dmtplib.h>
```

Inheritance diagram for oepdev::CAMM:



### Public Member Functions

- **CAMM** (psi::SharedWavefunction wfn, int n)
- virtual void [compute](#) (psi::SharedMatrix D, bool transition, int n)

*Compute DMTP's from the one-particle density matrix.*

### Additional Inherited Members

#### 17.8.1 Detailed Description

Cumulative Atomic Multipole Moments.

Cumulative atomic multipole representation of the molecular charge distribution. Method of Sokalski and Poirier. Ref.: W. A. Sokalski and R. A. Poirier, *Chem. Phys. Lett.*, 98(1) **1983**

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

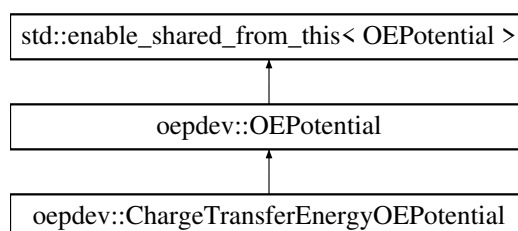
- oepdev/lib3d/[dmtplib.h](#)
- oepdev/lib3d/dmtplib\_camm.cc

## 17.9 oepdev::ChargeTransferEnergyOEPotential Class Reference

Generalized One-Electron Potential for Charge-Transfer Interaction Energy.

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

Inheritance diagram for oepdev::ChargeTransferEnergyOEPotential:



## Public Member Functions

- **ChargeTransferEnergyOEPotential** (SharedWavefunction [wfn](#), SharedBasisSet auxiliary, SharedBasisSet intermediate, Options &options)
- **ChargeTransferEnergyOEPotential** (SharedWavefunction [wfn](#), Options &options)
- virtual void [compute](#) (const std::string &oepType) override  
*Compute matrix forms of all OEP's within a specified OEP type.*
- virtual void [compute\\_3D](#) (const std::string &oepType, const double &x, const double &y, const double &z, std::shared\_ptr< psi::Vector > &v) override  
*Compute value of potential in point x, y, z and save at v.*
- virtual void [print\\_header](#) () const override

## Additional Inherited Members

### 17.9.1 Detailed Description

Generalized One-Electron Potential for Charge-Transfer Interaction Energy.

Contains the following OEP types:

- `Otto-Ladik.V1` - DF-based term
- `Otto-Ladik.V2` - ESP-based term
- `Otto-Ladik.V3` - ESP-based term

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

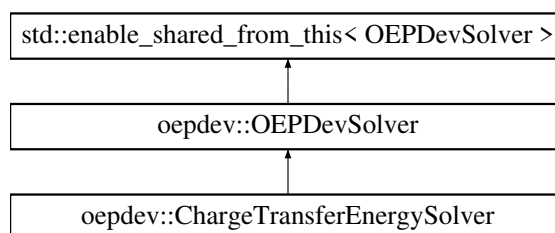
- `oepdev/liboep/oep.h`
- `oepdev/liboep/oep_energy_ct.cc`

## 17.10 oepdev::ChargeTransferEnergySolver Class Reference

Compute the Charge-Transfer interaction energy between unperturbed wavefunctions.

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

Inheritance diagram for oepdev::ChargeTransferEnergySolver:



## Public Member Functions

- **ChargeTransferEnergySolver** (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

### 17.10.1 Detailed Description

Compute the Charge-Transfer interaction energy between unperturbed wavefunctions.

The implemented methods are shown below

Keyword	Method Description
<b>Benchmark Methods</b>	
OTTO_LADIK	*Default*. CT energy at HF level from Otto and Ladik (1975).
EFP2	CT energy at HF level from EFP2 model.
<b>OEP-Based Methods</b>	
OTTO_LADIK	*Default*. OEP-based Otto-Ladik expressions.

Table 17.1: Methods available in the Solver

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

Below the detailed description of the implemented equations is given for each of the above provided methods. In the formulae across, it is assumed that the orbitals are real. The Coulomb notation for electron repulsion integrals (ERI's) is adopted; i.e.,

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

Greek subscripts denote basis set orbitals whereas Italic subscripts denote the occupied molecular orbitals.

The CT energy between molecules *A* and *B* is given by

$$E^{\text{CT}} = E^{A^+B^-} + E^{A^-B^+}$$

## Benchmark Methods

CT energy at HF level by Otto and Ladik (1975).

For a closed-shell system, CT energy equation of Otto and Ladik becomes

$$E^{A^+B^-} \approx 2 \sum_{i \in A}^{\text{Occ}_A} \sum_{n \in B}^{\text{Vir}_B} \frac{V_{in}^2}{\epsilon_i - \epsilon_n}$$

where

$$V_{in} = V_{in}^B + 2 \sum_{j \in B}^{\text{Occ}_B} (in|jj) - \sum_{k \in A}^{\text{Occ}_A} S_{kn} \left\{ V_{ik}^B + 2 \sum_{j \in B}^{\text{Occ}_B} (ik|jj) \right\} \\ - \sum_{j \in B}^{\text{Occ}_B} \left[ S_{ij} \left\{ V_{nj}^A + 2 \sum_{k \in A}^{\text{Occ}_A} (1 - \delta_{ik})(nj|kk) \right\} - (nj|ij) \right] + \sum_{k \in A}^{\text{Occ}_A} \sum_{j \in B}^{\text{Occ}_B} S_{kj} (1 - \delta_{ik})(ik|nj)$$

and analogously the twin term.

CT energy at HF level by EFP2.

In EFP2 method, CT energy is given as

$$E^{A+B-} \approx 2 \sum_{i \in A} \sum_{n \in B}^{\text{Occ}_A \text{Vir}_B} \frac{V_{in}^2}{F_{ii} - T_{nn}}$$

where

$$V_{in}^2 = \frac{V_{in}^B - \sum_{m \in A}^{\text{All}_A} V_{im} S_{mn}^B}{1 - \sum_{m \in A}^{\text{All}_A} S_{mn}^2} \left\{ V_{in}^B - \sum_{m \in A}^{\text{All}_A} V_{im}^B S_{mn} + \sum_{j \in B}^{\text{Occ}_B} S_{ij} \left( T_{nj} - \sum_{m \in A}^{\text{All}_A} S_{nm} T_{mj} \right) \right\}$$

and analogously the twin term.

## OEP-Based Methods

### OEP-Based Otto-Ladik's theory

After introducing OEP's, the original Otto-Ladik's theory is reformulated *without* approximation as

$$E^{A+B-} \approx 2 \sum_{i \in A} \sum_{n \in B}^{\text{Occ}_A \text{Vir}_B} \frac{\left( V_{in}^{\text{DF}} + V_{in}^{\text{ESP,A}} + V_{in}^{\text{ESP,B}} \right)^2}{\epsilon_i - \epsilon_n}$$

where

$$\begin{aligned} V_{in}^{\text{DF}} &= \sum_{\eta \in B}^{\text{Aux}_B} S_{i\eta} G_{\eta n}^B \\ V_{in}^{\text{ESP,A}} &= \sum_{k \in A}^{\text{Occ}_A} \sum_{j \in B}^{\text{Occ}_B} S_{kj} \sum_{x \in A} V_{nj}^{(x)} q_{ik}^{(x)} \\ V_{in}^{\text{ESP,B}} &= - \sum_{k \in A}^{\text{Occ}_A} S_{kn} V_{ik}^B \end{aligned}$$

The OEP matrix for density fitted part is given by

$$G_{\eta n}^B = \sum_{\eta' \in B}^{\text{Aux}_B} [S^{-1}]_{\eta\eta'} \left\{ V_{\eta'n}^B + \sum_{j \in B}^{\text{Occ}_B} [2(\eta'n|jj) - (\eta'j|nj)] \right\}$$

The OEP ESP-A charges are fit to reproduce the OEP potential

$$v_{ik}^A(\mathbf{r}) \equiv (1 - \delta_{ik}) \int \frac{\phi_i(\mathbf{r}') \phi_k(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' - \delta_{ik} \left( \sum_{x \in A} \frac{-Z_x}{|\mathbf{r} - \mathbf{r}_x|} + 2 \sum_{k \in A}^{\text{Occ}_A} \int \frac{\phi_k(\mathbf{r}') \phi_k(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' - 2 \int \frac{\phi_i(\mathbf{r}') \phi_i(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' \right)$$

so that

$$v_{ik}^A(\mathbf{r}) \cong \sum_{x \in A} \frac{q_{ik}^{(x)}}{|\mathbf{r} - \mathbf{r}_x|}$$

The OEP ESP-B charges are fit to reproduce the electrostatic potential of molecule *B* (they are standard ESP charges).

## 17.10.2 Member Function Documentation

**17.10.2.1** `double ChargeTransferEnergySolver::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](#).

17.10.2.2 `double ChargeTransferEnergySolver::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/libsolver/solver.h](#)
- [oepdev/libsolver/solver\\_energy\\_ct.cc](#)

## 17.11 oepdev::CPHF Class Reference

[CPHF](#) solver class.

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

### Public Member Functions

- [CPHF](#) (SharedWavefunction ref\_wfn, Options &[options](#))  
*Constructor.*
- [~CPHF](#) ()  
*Destructor.*
- void [compute](#) (void)  
*run the calculations*
- void [print](#) (void) const  
*print to output file*
- int [nocc](#) (void) const  
*get the number of occupied orbitals*
- std::shared\_ptr< Wavefunction > [wfn](#) (void) const  
*grab the wavefunction*
- Options & [options](#) (void) const  
*grab the Psi4 options*
- std::shared\_ptr< Matrix > [polarizability](#) (void) const  
*retrieve the molecular (total) polarizability*
- std::shared\_ptr< Matrix > [polarizability](#) (int i) const  
*retrieve the i-th orbital-associated polarizability*
- std::shared\_ptr< Matrix > [polarizability](#) (int i, int j) const  
*retrieve the charge-transfer polarizability associated with orbitals i and j*
- std::shared\_ptr< Matrix > [X](#) (int x) const  
*retrieve the X operator O-V perturbation matrix in AO basis for x-th component*

- `std::vector< std::shared_ptr< Matrix > > X` (void) const  
*retrieve the X operator O-V perturbation matrix in AO basis for all three Cartesian components*
- `std::shared_ptr< Matrix > X_mo` (int x) const  
*retrieve the X operator O-V perturbation matrix in MO basis for x-th component*
- `std::vector< std::shared_ptr< Matrix > > X_mo` (void) const  
*retrieve the X operator O-V perturbation matrix in MO basis for all three Cartesian components*
- `std::shared_ptr< Matrix > F_mo` (int x) const  
*retrieve the F operator O-V perturbation matrix in MO basis for x-th component*
- `std::vector< std::shared_ptr< Matrix > > F_mo` (void) const  
*retrieve the F operator O-V perturbation matrix in MO basis for all three Cartesian components*
- `std::shared_ptr< Matrix > T` (void) const  
*retrieve the transformation from old to new MO's*
- `std::shared_ptr< Matrix > Cocc` (void) const  
*retrieve the Cocc*
- `std::shared_ptr< Matrix > Cvir` (void) const  
*retrieve the Cvir*
- `std::shared_ptr< Vector > lmo_centroid` (int i) const  
*retrieve the i-th orbital (LMO) centroid*
- `std::shared_ptr< Localizer > localizer` (void) const  
*retrieve the orbital localizer*

## Protected Attributes

- `std::shared_ptr< psi::Wavefunction > _wfn`  
*Wavefunction object.*
- `std::shared_ptr< Localizer > _localizer`  
*Orbital localizer.*
- `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 > _molecularPolarizability`  
*Total (molecular) polarizability tensor.*
- `std::vector< std::shared_ptr  
< Vector > > _orbitalCentroids`  
*LMO centroids.*
- `std::vector< std::shared_ptr  
< Matrix > > _orbitalPolarizabilities`  
*orbital-associated polarizability tensors*
- `std::vector< std::vector  
< std::shared_ptr< Matrix > > > _orbitalChargeTransferPolarizabilities`  
*orbital-orbital charge-transfer polarizability tensors*
- `std::vector< std::shared_ptr  
< Matrix > > _X_OV_ao_matrices`  
*Perturbation X Operator O->V matrices in AO basis.*
- `std::vector< std::shared_ptr  
< Matrix > > _X_OV_mo_matrices`  
*Perturbation X Operator O->V matrices in MO basis.*
- `std::vector< std::shared_ptr  
< Matrix > > _F_OV_mo_matrices`  
*Electric Field Operator O->V matrices in MO basis.*
- `std::shared_ptr< psi::Matrix > _T`  
*Transformation from old to new MO's.*

### 17.11.1 Detailed Description

**CPHF** solver class.

Solves **CPHF** equations (now only for RHF wavefunction). Computes molecular and polarizabilities associated with the localized molecular orbitals (LMO).

#### Note

Useful options:

- `CPHF_CONVER` - convergence of **CPHF**. Default: `1e-8 (au)`
- `CPHF_CONVER` - maximum number of iterations. Default: `50`
- `CPHF_DIIS` - wheather use DIIS or not. Default: `true`
- `CPHF_DIIS_DIM` - dimension of iterative subspace. Default: `3`
- `CPHF_LOCALIZE` - localize the molecular orbitals? Default: `true`
- `CPHF_LOCALIZER` - set orbital localization method. Available: `BOYS` and `PIPEK_MEZEY`. Default: `BOYS`

## 17.11.2 Constructor & Destructor Documentation

### 17.11.2.1 oepdev::CPHF::CPHF ( SharedWavefunction *ref\_wfn*, Options & *options* )

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:

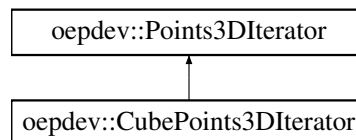
- oepdev/libutil/cphf.h
- oepdev/libutil/cphf.cc

## 17.12 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

#### 17.12.1 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:

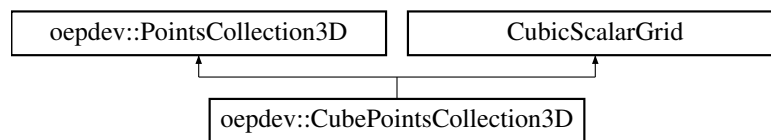
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

## 17.13 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, const int &col=0)

### Additional Inherited Members

#### 17.13.1 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/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

## 17.14 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)

### 17.14.1 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.

### 17.14.2 Constructor & Destructor Documentation

#### 17.14.2.1 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

### 17.14.3 Member Function Documentation

#### 17.14.3.1 void oepdev::DIISManager::compute ( void )

Perform DIIS interpolation.

#### 17.14.3.2 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

#### 17.14.3.3 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:

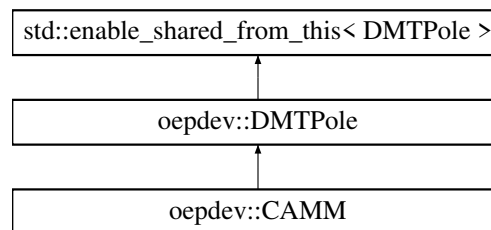
- oepdev/libutil/[diis.h](#)
- oepdev/libutil/diis.cc

## 17.15 oepdev::DMTPole Class Reference

Distributed Multipole Analysis Container and Computer. Abstract Base.

```
#include <dmt.h>
```

Inheritance diagram for oepdev::DMTPole:



### Public Member Functions

- virtual [~DMTPole](#) ()  
*Destructor.*
- virtual bool [has\\_charges](#) () const  
*Has distributed charges?*
- virtual bool [has\\_dipoles](#) () const  
*Has distributed dipoles?*
- virtual bool [has\\_quadrupoles](#) () const  
*Has distributed quadrupoles?*
- virtual bool [has\\_octupoles](#) () const  
*Has distributed octupoles?*
- virtual bool [has\\_hexadecapoles](#) () const  
*Has distributed hexadecapoles?*
- virtual psi::SharedMatrix [centres](#) () const  
*Get the positions of distribution centres.*
- virtual psi::SharedMatrix [origins](#) () const  
*Get the positions of distribution origins.*
- virtual std::vector  
  < psi::SharedMatrix > [charges](#) () const  
*Get the distributed charges.*
- virtual std::vector  
  < psi::SharedMatrix > [dipoles](#) () const  
*Get the distributed dipoles.*
- virtual std::vector  
  < psi::SharedMatrix > [quadrupoles](#) () const  
*Get the distributed quadrupoles.*
- virtual std::vector  
  < psi::SharedMatrix > [octupoles](#) () const  
*Get the distributed octupoles.*
- virtual std::vector  
  < psi::SharedMatrix > [hexadecapoles](#) () const  
*Get the distributed hexadecapoles.*
- virtual psi::SharedMatrix [charges](#) (int i) const  
*Get the distributed charges for the i-th distribution.*
- virtual psi::SharedMatrix [dipoles](#) (int i) const

- Get the distributed dipoles for the  $i$ th distribution.*

  - virtual psi::SharedMatrix [quadrupoles](#) (int i) const

*Get the distributed quadrupoles for the  $i$ th distribution.*

  - virtual psi::SharedMatrix [octupoles](#) (int i) const

*Get the distributed octupoles for the  $i$ th distribution.*

  - virtual psi::SharedMatrix [hexadecapoles](#) (int i) const

*Get the distributed hexadecapoles for the  $i$ th distribution.*

  - virtual int [n\\_sites](#) () const

*Get the number of distributed sites.*

  - virtual int [n\\_dmtp](#) () const

*Get the number of distributions.*

  - void [set\\_charges](#) (std::vector< psi::SharedMatrix > M)

*Set the distributed charges.*

  - void [set\\_dipoles](#) (std::vector< psi::SharedMatrix > M)

*Set the distributed dipoles.*

  - void [set\\_quadrupoles](#) (std::vector< psi::SharedMatrix > M)

*Set the distributed quadrupoles.*

  - void [set\\_octupoles](#) (std::vector< psi::SharedMatrix > M)

*Set the distributed octupoles.*

  - void [set\\_hexadecapoles](#) (std::vector< psi::SharedMatrix > M)

*Set the distributed hexadecapoles.*

  - void [set\\_charges](#) (psi::SharedMatrix M, int i)

*Set the distributed charges for the  $i$ th distribution.*

  - void [set\\_dipoles](#) (psi::SharedMatrix M, int i)

*Set the distributed dipoles for the  $i$ th distribution.*

  - void [set\\_quadrupoles](#) (psi::SharedMatrix M, int i)

*Set the distributed quadrupoles for the  $i$ th distribution.*

  - void [set\\_octupoles](#) (psi::SharedMatrix M, int i)

*Set the distributed octupoles for the  $i$ th distribution.*

  - void [set\\_hexadecapoles](#) (psi::SharedMatrix M, int i)

*Set the distributed hexadecapoles for the  $i$ th distribution.*

  - virtual void [recenter](#) (psi::SharedMatrix new\_origins, int i)

*Change origins of the distributed multipole moments of  $i$ th set.*

  - virtual void [recenter](#) (psi::SharedMatrix new\_origins)

*Change origins of the distributed multipole moments of all sets.*

  - virtual void [compute](#) (psi::SharedMatrix D, bool transition, int i)=0

*Compute DMTP's from the one-particle density matrix.*

  - void [compute](#) (std::vector< psi::SharedMatrix > D, std::vector< bool > transition)

*Compute DMTP's from the set of the one-particle density matrices.*

  - void [compute](#) (void)

*Compute DMTP's from the sum of the ground-state alpha and beta one-particle density matrices (transition=false, i=0)*

  - std::shared\_ptr< [MultipoleConvergence](#) > [energy](#) (std::shared\_ptr< [DMTPole](#) > other, MultipoleConvergence::ConvergenceLevel max\_clevel=MultipoleConvergence::R5)

*Evaluate the generalized interaction energy.*

  - std::shared\_ptr< [MultipoleConvergence](#) > [potential](#) (std::shared\_ptr< [DMTPole](#) > other, MultipoleConvergence::ConvergenceLevel max\_clevel=MultipoleConvergence::R5)

*Evaluate the generalized potential.*

## Static Public Member Functions

- static std::shared\_ptr< [DMTPole](#) > [build](#) (const std::string &type, std::shared\_ptr< psi::Wavefunction > wfn, int n=1)  
*Build an empty DMTP object from the wavefunction.*

## Protected Member Functions

- [DMTPole](#) (std::shared\_ptr< psi::Wavefunction > wfn, int n)  
*Construct an empty DMTP object from the wavefunction.*
- void [compute\\_integrals](#) ()  
*Compute multipole integrals.*
- void [compute\\_order](#) ()  
*Compute maximum order of the integrals.*
- virtual void [allocate](#) ()  
*Initialize and allocate memory.*

## Protected Attributes

- std::string [name\\_](#)  
*Name of the distribution method.*
- psi::SharedMolecule [mol\\_](#)  
*Molecule associated with this DMTP.*
- psi::SharedWavefunction [wfn\\_](#)  
*Wavefunction associated with this DMTP.*
- psi::SharedBasisSet [primary\\_](#)  
*Basis set (primary)*
- int [nDMTPs\\_](#)  
*Number of DMTP's.*
- int [nSites\\_](#)  
*Number of DMTP sites.*
- int [order\\_](#)  
*Maximum order of the multipole.*
- std::vector< psi::SharedMatrix > [mplnts\\_](#)  
*Multipole integrals.*
- bool [hasCharges\\_](#)  
*Has distributed charges?*
- bool [hasDipoles\\_](#)  
*Has distributed dipoles?*
- bool [hasQuadrupoles\\_](#)  
*Has distributed quadrupoles?*
- bool [hasOctupoles\\_](#)  
*Has distributed octupoles?*
- bool [hasHexadecapoles\\_](#)  
*Has distributed hexadecapoles?*
- psi::SharedMatrix [centres\\_](#)  
*DMTP centres.*
- psi::SharedMatrix [origins\\_](#)  
*DMTP origins.*
- std::vector< psi::SharedMatrix > [charges\\_](#)  
*DMTP charges.*

- `std::vector< psi::SharedMatrix > dipoles_`  
*DMTP dipoles.*
- `std::vector< psi::SharedMatrix > quadrupoles_`  
*DMTP quadrupoles.*
- `std::vector< psi::SharedMatrix > octupoles_`  
*DMTP octupoles.*
- `std::vector< psi::SharedMatrix > hexadecapoles_`  
*DMTP hexadecapoles.*

## Friends

- class **MultipoleConvergence**

### 17.15.1 Detailed Description

Distributed Multipole Analysis Container and Computer. Abstract Base.

Handles the distributed multipole expansions up to hexadecapole.

### 17.15.2 Constructor & Destructor Documentation

17.15.2.1 `oepdev::DMTPole::DMTPole ( std::shared_ptr< psi::Wavefunction > wfn, int n )` [protected]

Construct an empty DMTP object from the wavefunction.

Parameters

<i>wfn</i>	- wavefunction
<i>n</i>	- number of DMTP sets Do not use this constructor. Use the <a href="#">DMTPole::build</a> method.

### 17.15.3 Member Function Documentation

17.15.3.1 `std::shared_ptr< DMTPole > oepdev::DMTPole::build ( const std::string & type, std::shared_ptr< psi::Wavefunction > wfn, int n = 1 )` [static]

Build an empty DMTP object from the wavefunction.

Parameters

<i>type</i>	- DMTP method. Available: <a href="#">Camm</a> .
<i>wfn</i>	- wavefunction
<i>n</i>	- number of DMTP sets

Returns

DMTP distribution

17.15.3.2 `std::shared_ptr< MultipoleConvergence > oepdev::DMTPole::energy ( std::shared_ptr< DMTPole > other, MultipoleConvergence::ConvergenceLevel max_clevel = MultipoleConvergence::R5 )`

Evaluate the generalized interaction energy.

## Parameters

<i>other</i>	- interacting DMTP distribution.
<i>max_clevel</i>	- maximum convergence level (see below).

## Returns

The generalized interaction energy convergence (A.U. units)

The following convergence levels are available:

- `MultipoleConvergence::R1`: includes qq terms.
- `MultipoleConvergence::R2`: includes dq terms and above.
- `MultipoleConvergence::R3`: includes qQ, dd terms and above.
- `MultipoleConvergence::R4`: includes qO, dQ terms and above.
- `MultipoleConvergence::R5`: includes qH, dO, QQ terms and above.

**17.15.3.3** `std::shared_ptr< MultipoleConvergence > oepdev::DMTPole::potential ( std::shared_ptr< DMTPole > other, MultipoleConvergence::ConvergenceLevel max_clevel = MultipoleConvergence::R5 )`

Evaluate the generalized potential.

## Parameters

<i>other</i>	- interacting DMTP distribution.
<i>max_clevel</i>	- maximum convergence level (see below).

## Returns

The generalized potential convergence (A.U. units)

The following convergence levels are available:

- `MultipoleConvergence::R1`: includes qq terms.
- `MultipoleConvergence::R2`: includes dq terms and above.
- `MultipoleConvergence::R3`: includes qQ, dd terms and above.
- `MultipoleConvergence::R4`: includes qO, dQ terms and above.
- `MultipoleConvergence::R5`: includes qH, dO, QQ terms and above.

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

- [oepdev/lib3d/dmtp.h](#)
- [oepdev/lib3d/dmtp\\_base.cc](#)

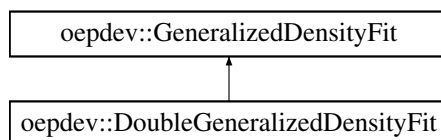
## 17.16 oepdev::DoubleGeneralizedDensityFit Class Reference

Generalized Density Fitting Scheme - Double Fit.

```
#include <oep_gdf.h>
```

Inheritance diagram for `oepdev::DoubleGeneralizedDensityFit`:





## Public Member Functions

- **DoubleGeneralizedDensityFit** (std::shared\_ptr< psi::BasisSet > bs\_auxiliary, std::shared\_ptr< psi::BasisSet > bs\_intermediate, std::shared\_ptr< psi::Matrix > v\_vector)
- std::shared\_ptr< psi::Matrix > **compute** (void)

*Perform the generalized density fit.*

## Additional Inherited Members

### 17.16.1 Detailed Description

Generalized Density Fitting Scheme - Double Fit.

The density fitting map projects the OEP onto an arbitrary (not necessarily complete) auxiliary basis set space through application of the self energy minimization technique. The resulting three-electron repulsion integrals are computed by utilizing the resolution of identity in an intermediate, nearly-complete basis set space, hence performing an internal density fitting in nearly complete basis. Refer to [density fitting specialized for OEP's](#) for more details.

### 17.16.2 Determination of the OEP matrix

Coefficients **G** are computed by using the following relation

$$\mathbf{G} = \mathbf{A}^{-1} \cdot \mathbf{R} \cdot \mathbf{H}$$

where the intermediate projection matrix is given by

$$\mathbf{H} = \mathbf{S}^{-1} \cdot \mathbf{V}$$

In the above equations,

$$\begin{aligned} A_{\xi\xi'} &= (\xi||\xi') \\ R_{\xi\varepsilon} &= (\xi||\varepsilon) \\ S_{\varepsilon\varepsilon'} &= (\varepsilon|\varepsilon') \\ V^{\varepsilon i} &= (\varepsilon|\hat{v}i) \end{aligned}$$

The following labeling convention is used here:

- $i$  denotes the arbitrary state vector
- $\xi$  denotes the auxiliary basis set element
- $\varepsilon$  denotes the intermediate (nearly complete) basis set element

In the above,  $|$  denotes the single integration over electron coordinate, i.e.,

$$(a|b) \equiv \int d\mathbf{r} \phi_a^*(\mathbf{r}) \phi_b(\mathbf{r})$$

whereas  $||$  acts as shown below:

$$(a||b) \equiv \iint d\mathbf{r}' d\mathbf{r}'' \frac{\phi_a^*(\mathbf{r}') \phi_b(\mathbf{r}'')}{|\mathbf{r}' - \mathbf{r}''|}$$

The spatial form of the potential operator  $\hat{v}$  can be expressed by

$$v(\mathbf{r}) \equiv \int d\mathbf{r}' \frac{\rho(\mathbf{r}')}{|\mathbf{r}' - \mathbf{r}|}$$

with  $\rho(\mathbf{r})$  being the effective one-electron density associated with  $\hat{v}$ .

### 17.16.2.1 Theory behind the double GDF scheme

In order to perform the generalized density fitting in an incomplete auxiliary basis set, one must apply the following formula:

$$\mathbf{G} = \mathbf{A}^{-1} \cdot \mathbf{B}$$

where one encounters the need of evaluation of the following *three-electron integrals*

$$B_{\xi i} = (\xi || \hat{v} i) \equiv \iiint d\mathbf{r}' d\mathbf{r}'' d\mathbf{r}''' \phi_{\xi}^*(\mathbf{r}') \frac{1}{|\mathbf{r}' - \mathbf{r}''|} \rho(\mathbf{r}''') \frac{1}{|\mathbf{r}''' - \mathbf{r}''|} \phi_i(\mathbf{r}'')$$

Computation of all the necessary integrals of this kind is very costly and impractical for larger molecules. However, one can use the same trick that is a kernel of the OEP technique introduced in the OEPDev project, i.e., introduce the effective potential in order to get rid of one integration. This can be done by performing the generalized density fitting in the nearly complete intermediate basis

$$\hat{v}|i) \cong \sum_{\varepsilon} H_{\varepsilon i} |\varepsilon)$$

Note that this is done just for the sake of factorizing the triple integral and computing the OEP matrix for the incomplete auxiliary basis. Therefore, the intermediate basis set is used just for a while during density fitting and is no longer necessary later on. By inserting the above identity to the triple integral one can transform it into a sum of the two-electron integrals that are much easier to evaluate. This leads to equations given in the beginning of this section.

## 17.16.3 Member Function Documentation

17.16.3.1 `std::shared_ptr< psi::Matrix > DoubleGeneralizedDensityFit::compute ( void ) [virtual]`

Perform the generalized density fit.

### Returns

The OEP coefficients  $G_{\xi i}$

Implements [oepdev::GeneralizedDensityFit](#).

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

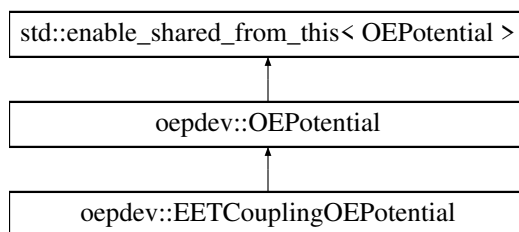
- [oepdev/liboep/oep\\_gdf.h](#)
- [oepdev/liboep/oep\\_gdf.cc](#)

## 17.17 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, SharedBasisSet intermediate, Options &options)
- **EETCouplingOEPotential** (SharedWavefunction [wfn](#), Options &options)
- virtual void [compute](#) (const std::string &oepType) override  
*Compute matrix forms of all OEP's within a specified OEP type.*
- virtual void [compute\\_3D](#) (const std::string &oepType, const double &x, const double &y, const double &z, std::shared\_ptr< psi::Vector > &v) override  
*Compute value of potential in point x, y, z and save at v.*
- virtual void **print\_header** () const override

## Additional Inherited Members

### 17.17.1 Detailed Description

Generalized One-Electron Potential for EET coupling calculations.

Contains the following OEP types:

- Fujimoto.ET1
- Fujimoto.ET2
- Fujimoto.HT1
- Fujimoto.HT1
- Fujimoto.HT2
- Fujimoto.CT1
- Fujimoto.CT2

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

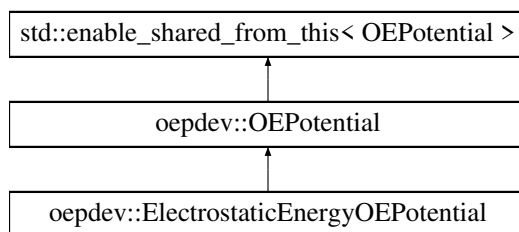
- oepdev/liboep/[oep.h](#)
- oepdev/liboep/oep\_coupling\_eet.cc

## 17.18 oepdev::ElectrostaticEnergyOEPotential Class Reference

Generalized One-Electron Potential for Electrostatic Energy.

```
#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  
*Compute matrix forms of all OEP's within a specified OEP type.*
- virtual void [compute\\_3D](#) (const std::string &oepType, const double &x, const double &y, const double &z, std::shared\_ptr< psi::Vector > &v) override  
*Compute value of potential in point x, y, z and save at v.*
- virtual void [print\\_header](#) () const override

## Additional Inherited Members

### 17.18.1 Detailed Description

Generalized One-Electron Potential for Electrostatic Energy.

Contains the following OEP types:

- $V$

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

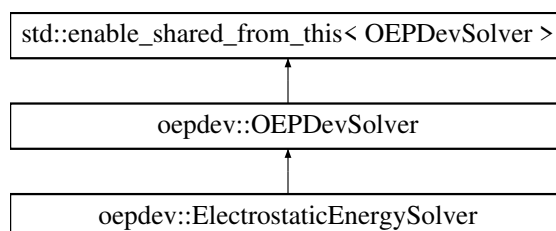
- [oepdev/liboep/oep.h](#)
- [oepdev/liboep/oep\\_energy\\_coul.cc](#)

## 17.19 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

### 17.19.1 Detailed Description

Compute the Coulombic interaction energy between unperturbed wavefunctions.

The implemented methods are shown in below

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

Table 17.2: 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

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

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

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

and the electron-electron repulsion energy is

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

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

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

and the electron-electron repulsion energy is

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

## 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,

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

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.

## 17.19.2 Member Function Documentation

17.19.2.1 `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](#).

```
17.19.2.2 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:

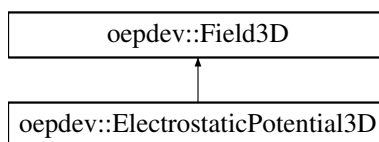
- [oepdev/libsolver/solver.h](#)
- [oepdev/libsolver/solver\\_energy\\_coul.cc](#)

## 17.20 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 std::shared\_ptr< psi::Vector > [compute\\_xyz](#) (const double &x, const double &y, const double &z)  
*Compute a value of 3D field at point (x, y, z)*
- virtual void [print](#) () const  
*Print information of the object to Psi4 output.*

### Additional Inherited Members

#### 17.20.1 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  $\omega$  electron spin, and  $V_{\mu\nu}(\mathbf{r})$  is the potential one-electron integral defined by

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

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

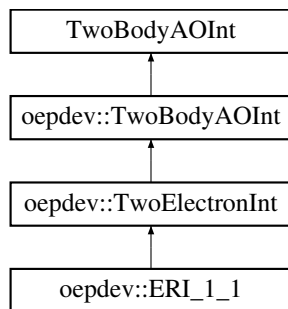
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

## 17.21 oepdev::ERI\_1\_1 Class Reference

2-centre ERI of the form (a|O(2)|b) where  $O(2) = 1/r_{12}$ .

```
#include <eri.h>
```

Inheritance diagram for oepdev::ERI\_1\_1:



### Public Member Functions

- [ERI\\_1\\_1](#) (const [IntegralFactory](#) \*integral, int deriv=0, bool use\_shell\_pairs=false)  
*Constructor. Use [oepdev::IntegralFactory](#) to generate this object.*
- [~ERI\\_1\\_1](#) ()  
*Destructor.*

### Protected Member Functions

- `size_t` [compute\\_doublet](#) (int, int)  
*Compute ERI's between 2 shells.*



## Protected Attributes

- double \* `mdh_buffer_1_`  
Buffer for McMurchie-Davidson-Hermite coefficients for monomial expansion (shell 1)
- double \* `mdh_buffer_2_`  
Buffer for McMurchie-Davidson-Hermite coefficients for monomial expansion (shell 2)

### 17.21.1 Detailed Description

2-centre ERI of the form  $(a|O(2)|b)$  where  $O(2) = 1/r12$ .

ERI's are computed for a shell doublet  $(P|Q)$  and stored in the `target_full_` buffer, accessible through `buffer()` method:

$$\begin{aligned} &\text{For each } (n_1, l_1, m_1) \in P : \\ &\quad \text{For each } (n_2, l_2, m_2) \in Q : \\ &\quad \text{ERI} = (A|B)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] \end{aligned}$$

For detailed description of the McMurchie-Davidson scheme, refer to [The Integral Package Library](#).

### 17.21.2 Implementation

A set of ERI's in a shell is decontracted as

$$(A|B)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] = \sum_{ij} c_i(\alpha_1) c_j(\alpha_2) (i|j)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}]$$

where the primitive ERI is given by

$$(i|j)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] = \sum_{N_1=0}^{n_1} \sum_{L_1=0}^{l_1} \sum_{M_1=0}^{m_1} \sum_{N_2=0}^{n_2} \sum_{L_2=0}^{l_2} \sum_{M_2=0}^{m_2} d_{N_1}^{n_1} d_{L_1}^{l_1} d_{M_1}^{m_1} d_{N_2}^{n_2} d_{L_2}^{l_2} d_{M_2}^{m_2} [N_1 L_1 M_1 | N_2 L_2 M_2]$$

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

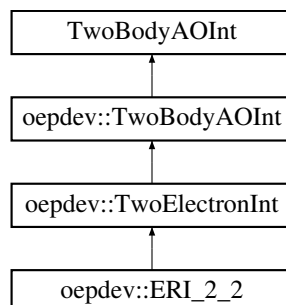
- `oepdev/libints/eri.h`
- `oepdev/libints/eri.cc`

## 17.22 oepdev::ERI\_2\_2 Class Reference

4-centre ERI of the form  $(ab|O(2)|cd)$  where  $O(2) = 1/r12$ .

```
#include <eri.h>
```

Inheritance diagram for `oepdev::ERI_2_2`:



## Public Member Functions

- [ERI\\_2\\_2](#) (const [IntegralFactory](#) \*integral, int deriv=0, bool use\_shell\_pairs=false)  
*Constructor. Use [oepdev::IntegralFactory](#) to generate this object.*
- [~ERI\\_2\\_2](#) ()  
*Destructor.*

## Protected Member Functions

- `size_t` [compute\\_quartet](#) (int, int, int, int)  
*Compute ERI's between 4 shells.*

## Protected Attributes

- `double *` [mdh\\_buffer\\_12\\_](#)  
*Buffer for McMurchie-Davidson-Hermite coefficients for binomial expansion (shells 1 and 2)*
- `double *` [mdh\\_buffer\\_34\\_](#)  
*Buffer for McMurchie-Davidson-Hermite coefficients for binomial expansion (shells 3 and 4)*

### 17.22.1 Detailed Description

4-centre ERI of the form (ab|O(2)|cd) where O(2) = 1/r<sup>12</sup>.

ERI's are computed for a shell quartet (PQ|RS) and stored in the `target_full_` buffer, accessible through `buffer()` method:

$$\begin{aligned} &\text{For each } (n_1, l_1, m_1) \in P : \\ &\quad \text{For each } (n_2, l_2, m_2) \in Q : \\ &\quad \quad \text{For each } (n_3, l_3, m_3) \in R : \\ &\quad \quad \quad \text{For each } (n_4, l_4, m_4) \in S : \\ &\quad \quad \quad \text{ERI} = (AB|CD)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] \end{aligned}$$

For detailed description of the McMurchie-Davidson scheme, refer to [The Integral Package Library](#).

### 17.22.2 Implementation

A set of ERI's in a shell is decontracted as

$$(AB|CD)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] = \sum_{ijkl} c_i(\alpha_1) c_j(\alpha_2) c_k(\alpha_3) c_l(\alpha_4) (ij|kl) [\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}]$$

where the primitive ERI is given by

$$\begin{aligned} (ij|kl) [\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] &= E_{ij}(\alpha_1, \alpha_2) E_{kl}(\alpha_3, \alpha_4) \\ &\times \sum_{N_1=0}^{n_1+n_2} \sum_{L_1=0}^{l_1+l_2} \sum_{M_1=0}^{m_1+m_2} \sum_{N_2=0}^{n_3+n_4} \sum_{L_2=0}^{l_3+l_4} \sum_{M_2=0}^{m_3+m_4} d_{N_1}^{n_1 n_2} d_{L_1}^{l_1 l_2} d_{M_1}^{m_1 m_2} d_{N_2}^{n_3 n_4} d_{L_2}^{l_3 l_4} d_{M_2}^{m_3 m_4} [N_1 L_1 M_1 | N_2 L_2 M_2] \end{aligned}$$

In the above equation, the multiplicative constants are given as

$$\begin{aligned} E_{ij}(\alpha_1, \alpha_2) &= \exp \left[ -\frac{\alpha_1 \alpha_2}{\alpha_1 + \alpha_2} |\mathbf{A} - \mathbf{B}|^2 \right] \\ E_{kl}(\alpha_3, \alpha_4) &= \exp \left[ -\frac{\alpha_3 \alpha_4}{\alpha_3 + \alpha_4} |\mathbf{C} - \mathbf{D}|^2 \right] \end{aligned}$$

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

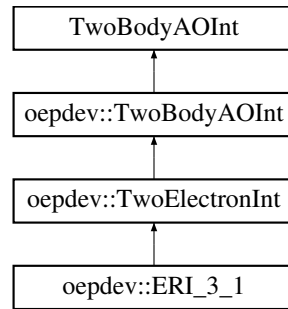
- [oepdev/libints/eri.h](#)
- [oepdev/libints/eri.cc](#)

## 17.23 oepdev::ERI\_3\_1 Class Reference

4-centre ERI of the form  $(abc|O(2)|d)$  where  $O(2) = 1/r_{12}$ .

```
#include <eri.h>
```

Inheritance diagram for oepdev::ERI\_3\_1:



### Public Member Functions

- [ERI\\_3\\_1](#) (const [IntegralFactory](#) \*integral, int deriv=0, bool use\_shell\_pairs=false)  
*Constructor. Use [oepdev::IntegralFactory](#) to generate this object.*
- [~ERI\\_3\\_1](#) ()  
*Destructor.*

### Protected Member Functions

- `size_t` [compute\\_quartet](#) (int, int, int, int)  
*Compute ERI's between 4 shells.*

### Protected Attributes

- `double *` [mdh\\_buffer\\_123\\_](#)  
*Buffer for McMurchie-Davidson-Hermite coefficients for trinomial expansion (shells 1, 2 and 3)*
- `double *` [mdh\\_buffer\\_4\\_](#)  
*Buffer for McMurchie-Davidson-Hermite coefficients for monomial expansion (shell 4)*

#### 17.23.1 Detailed Description

4-centre ERI of the form  $(abc|O(2)|d)$  where  $O(2) = 1/r_{12}$ .

ERI's are computed for a shell quartet  $(PQR|S)$  and stored in the `target_full_` buffer, accessible through `buffer()` method:

For each  $(n_1, l_1, m_1) \in P$  :  
 For each  $(n_2, l_2, m_2) \in Q$  :  
 For each  $(n_3, l_3, m_3) \in R$  :  
 For each  $(n_4, l_4, m_4) \in S$  :  

$$\text{ERI} = (ABC|D)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}]$$

For detailed description of the McMurchie-Davidson scheme, refer to [The Integral Package Library](#).

### 17.23.2 Implementation

A set of ERI's in a shell is decontracted as

$$(ABC|D)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] = \sum_{ijkl} c_i(\alpha_1) c_j(\alpha_2) c_k(\alpha_3) c_l(\alpha_4) (ijk|l) [\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}]$$

where the primitive ERI is given by

$$(ijk|l) [\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] = E_{ijk}(\alpha_1, \alpha_2, \alpha_3) \times \sum_{N_1=0}^{n_1+n_2+n_3} \sum_{L_1=0}^{l_1+l_2+l_3} \sum_{M_1=0}^{m_1+m_2+m_3} \sum_{N_2=0}^{n_4} \sum_{L_2=0}^{l_4} \sum_{M_2=0}^{m_4} d_{N_1}^{n_1 n_2 n_3} d_{L_1}^{l_1 l_2 l_3} d_{M_1}^{m_1 m_2 m_3} d_{N_2}^{n_4} d_{L_2}^{l_4} d_{M_2}^{m_4} [N_1 L_1 M_1 | N_2 L_2 M_2]$$

In the above equation, the multiplicative constants are given as

$$E_{ijk}(\alpha_1, \alpha_2, \alpha_3) = \exp \left[ -\frac{\alpha_1 \alpha_2}{\alpha_1 + \alpha_2} |\mathbf{A} - \mathbf{B}|^2 \right] \exp \left[ -\frac{(\alpha_1 + \alpha_2) \alpha_3}{\alpha_1 + \alpha_2 + \alpha_3} |\mathbf{P} - \mathbf{C}|^2 \right]$$

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

- [oepdev/libints/eri.h](#)
- [oepdev/libints/eri.cc](#)

## 17.24 oepdev::ESPSolver Class Reference

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

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

### Public Member Functions

- [ESPSolver](#) (SharedField3D field)  
*Construct from 3D vector field.*
- [ESPSolver](#) (SharedField3D field, psi::SharedMatrix [centres](#))  
*Construct from 3D vector field.*
- virtual [~ESPSolver](#) ()  
*Destructor.*
- virtual psi::SharedMatrix [charges](#) () const  
*Get the (fit) charges.*
- virtual psi::SharedMatrix [centres](#) () const  
*Get the charge distribution centres.*
- virtual void [set\\_charge\\_sums](#) (psi::SharedVector s)  
*Set the charge sums  $Q_p$ .*
- virtual void [set\\_charge\\_sums](#) (const double &s)  
*Set the charge sums  $Q_p$  (equal to all fields)*
- virtual void [compute](#) ()  
*Perform fitting of effective charges.*

## Protected Attributes

- const int `nCentres_`  
*Number of fit centres.*
- const int `nFields_`  
*Number of fields to fit.*
- SharedField3D `field_`  
*Scalar field.*
- psi::SharedMatrix `charges_`  
*Charges to be fit.*
- psi::SharedMatrix `centres_`  
*Centres, at which fit charges will reside.*
- psi::SharedVector `charge_sums_`  
*Vector of sums of partial charges.*

### 17.24.1 Detailed Description

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

Solves the least-squares problem to fit the generalized charges  $q_{m;p}$ , that reproduce the reference generalized potential  $v_p^{\text{ref}}(\mathbf{r})$  supplied by the `Field3D` object:

$$\int d\mathbf{r}' \left[ v_p^{\text{ref}}(\mathbf{r}') - \sum_m \frac{q_{m;p}}{|\mathbf{r}' - \mathbf{r}_m|} \right]^2 \rightarrow \text{minimize}$$

The charges are subject to the following constraint:

$$\sum_m q_{m;p} = Q_p \text{ for all } p$$

**Method description.**

$M$  generalized charges is found by solving the matrix equation

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

where the  $\mathbf{A}$  matrix of dimension  $(M+1) \times (M+1)$  and  $\mathbf{b}_p$  vector of length  $M+1$  are given as

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

$$b_{m;p} = \sum_i \frac{v_p^{\text{ref}}(\mathbf{r}_m)}{r_{im}}$$

In the above equation, summations run over all sample points, at which reference potential is known. The solution is stored in the  $M \times N$  matrix, where  $N$  is the dimensionality of the 3D vector field (i.e., the number of potentials supplied,  $p_{\text{max}}$ ). As a default,  $Q_p = 0$  for all potentials. This can be set by `oepdev::ESPSolver::set_charge_sums` method.

**Note**

Useful options:

- `ESP_PAD_SPHERE` - Padding spherical radius for random points selection. Default: 10.0 [A.U.]
- `ESP_NPOINTS_PER_ATOM` - Number of random points per atom in a molecule. Default: 1500
- `ESP_VDW_RADIUS_C` - The vdW radius for carbon atom. Default: 3.0 [A.U.]
- `ESP_VDW_RADIUS_H` - The vdW radius for hydrogen atom. Default: 4.0 [A.U.]
- `ESP_VDW_RADIUS_N` - The vdW radius for nitrogen atom. Default: 2.4 [A.U.]
- `ESP_VDW_RADIUS_O` - The vdW radius for oxygen atom. Default: 5.6 [A.U.]
- `ESP_VDW_RADIUS_F` - The vdW radius for fluorine atom. Default: 2.3 [A.U.]
- `ESP_VDW_RADIUS_CL` - The vdW radius for chlorine atom. Default: 2.9 [A.U.]

## 17.24.2 Constructor & Destructor Documentation

### 17.24.2.1 `oepdev::ESPSolver::ESPSolver ( SharedField3D field )`

Construct from 3D vector field.

Assume that the centres are on atoms associated with the 3D vector field.

Parameters

<i>field</i>	- oepdev 3D vector field object
--------------	---------------------------------

### 17.24.2.2 `oepdev::ESPSolver::ESPSolver ( SharedField3D field, psi::SharedMatrix centres )`

Construct from 3D vector field.

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

Parameters

<i>field</i>	- oepdev 3D vector field object
<i>centres</i>	- matrix with coordinates of charge distribution centres

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

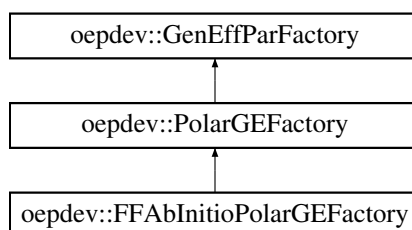
- [oepdev/lib3d/esp.h](#)
- [oepdev/lib3d/esp.cc](#)

## 17.25 `oepdev::FFAbInitioPolarGEFactory` Class Reference

Polarization GEFP Factory from First Principles: Finite-Difference Model. Arbitrary level of theory.

```
#include <gefp.h>
```

Inheritance diagram for `oepdev::FFAbInitioPolarGEFactory`:



### Public Member Functions

- **FFAbInitioPolarGEFactory** (`std::shared_ptr< psi::Wavefunction > wfn`, `psi::Options &opt`)
- virtual `std::shared_ptr< GenEffPar > compute` (void)  
*Compute the density matrix susceptibility tensors.*

### Additional Inherited Members

#### 17.25.1 Detailed Description

Polarization GEFP Factory from First Principles: Finite-Difference Model. Arbitrary level of theory.

Implements creation of the density matrix susceptibility tensors. Does not guarantee the idempotency of the density matrix in LCAO-MO variation, but for weak electric fields the idempotency is to be expected up to first order. The density matrix susceptibility tensor is represented by:

$$\delta D_{\alpha\beta} = \mathbf{B}_{\alpha\beta}^{(1)} \cdot \mathbf{F} + \mathbf{B}_{\alpha\beta}^{(2)} : \mathbf{F} \otimes \mathbf{F}$$

where  $\mathbf{B}_{\alpha\beta}^{(1)}$  is the density matrix dipole polarizability defined as

$$\mathbf{B}_{\alpha\beta}^{(1)} = \left. \frac{\partial D_{\alpha\beta}}{\partial \mathbf{F}} \right|_{\mathbf{F}=\mathbf{0}}$$

whereas  $\mathbf{B}_{\alpha\beta}^{(2)}$  is the density matrix dipole-dipole hyperpolarizability,

$$\mathbf{B}_{\alpha\beta}^{(2)} = \frac{1}{2} \left. \frac{\partial^2 D_{\alpha\beta}}{\partial \mathbf{F} \otimes \partial \mathbf{F}} \right|_{\mathbf{F}=\mathbf{0}}$$

The first derivative is evaluated numerically from central finite-field 3-point formula,

$$f' = \frac{f(h) - f(-h)}{2h} + \mathcal{O}(h^2)$$

where  $h$  is the differentiation step. Second derivatives are evaluated from the following formulae:

$$f_{uu} = \frac{f(h) + f(-h) - 2f(0)}{h^2} + \mathcal{O}(h^2)$$

$$f_{uw} = \frac{f(h, h) + f(-h, -h) + 2f(0) - f(h, 0) - f(-h, 0) - f(0, h) - f(0, -h)}{2h^2} + \mathcal{O}(h^2)$$

As long as the second-order susceptibility is considered, this susceptibility model works well for uniform weak, moderate and strong electric fields.

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

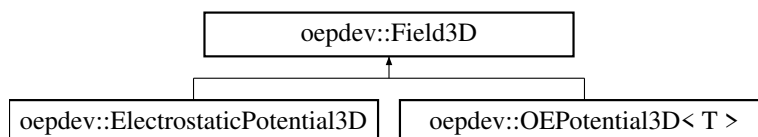
- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp\_polar\_ffabinitio.cc

## 17.26 oepdev::Field3D Class Reference

General Vector Field in 3D Space. Abstract base.

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

Inheritance diagram for oepdev::Field3D:



### Public Member Functions

- [Field3D](#) (const int &ndim, const int &np, const double &pad, psi::SharedWavefunction [wfn](#), psi::Options &options)

*Construct potential on random grid by providing wavefunction. Excludes space within vdW volume.*

- [Field3D](#) (const int &ndim, 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 `~Field3D ()`  
*Destructor.*
  - virtual `int npoints () const`  
*Get the number of points at which the 3D 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  $\{ [x, y, z, f_1(x, y, z), f_2(x, y, z), \dots, f_n(x, y, z)] \}$  where  $n = ndim$ .*
  - 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.*
  - `int dimension () const`  
*Get the number of fields.*
  - virtual `void compute ()`  
*Compute the 3D field in each point from the point collection.*
  - virtual `std::shared_ptr< psi::Vector > compute_xyz (const double &x, const double &y, const double &z)=0`  
*Compute a value of 3D 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< Field3D > build (const std::string &type, const int &np, const double &pad, psi::SharedWavefunction wfn, psi::Options &options, const int &ndim=1)`  
*Build 3D field of random points. vdW volume is excluded.*
- static `shared_ptr< Field3D > 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, const int &ndim=1)`  
*Build 3D field of points on a g09-cube grid.*

## Protected Attributes

- `std::shared_ptr< PointsCollection3D > pointsCollection_`  
*Collection of points at which the 3D field is to be computed.*
- `std::shared_ptr< psi::Matrix > data_`  
*The data matrix in a form  $\{ [x, y, z, f_1(x, y, z), f_2(x, y, z), \dots, f_n(x, y, z)] \}$  where  $n = nDim_$ .*
- `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.*
- `int nDim_`  
*Dimensionality of the 3D field (1: scalar field, 2>: vector field)*
- `bool isComputed_`  
*Has data already computed?*

### 17.26.1 Detailed Description

General Vector Field in 3D Space. Abstract base.

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

- Electrostatic potential - computes electrostatic potential (requires wavefunction)
- Template of generic classes - compute custom vector 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 3D vector fields are currently implemented:

- ELECTROSTATIC POTENTIAL

### 17.26.2 Constructor & Destructor Documentation

17.26.2.1 `oepdev::Field3D::Field3D ( const int & ndim, 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.

Construct potential on random grid by providing molecule. Excludes space within vdW volume `Field3D(const int& ndim, const int& np, const double& pad, psi::SharedMolecule mol, psi::Options& options);`

### 17.26.3 Member Function Documentation

17.26.3.1 `std::shared_ptr< Field3D > oepdev::Field3D::build ( const std::string & type, const int & np, const double & pad, psi::SharedWavefunction wfn, psi::Options & options, const int & ndim = 1 ) [static]`

Build 3D field of random points. vdW volume is excluded.

## Parameters

<i>ndim</i>	- dimensionality of 3D field (1: scalar field, >2: vector field)
<i>type</i>	- type of 3D 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

17.26.3.2 `std::shared_ptr< Field3D > oepdev::Field3D::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, const int & ndim = 1 ) [static]`

Build 3D field of points on a g09-cube grid.

## Parameters

<i>ndim</i>	- dimensionality of 3D field (1: scalar field, >2: vector field)
<i>type</i>	- type of 3D 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/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

## 17.27 oepdev::Fourier9 Struct Reference

Simple structure to hold the Fourier series expansion coefficients for  $N=4$ .

```
#include <unitary_optimizer.h>
```

## Public Attributes

- double **a0**
- double **a1**
- double **a2**
- double **a3**
- double **a4**
- double **b1**
- double **b2**
- double **b3**
- double **b4**

### 17.27.1 Detailed Description

Simple structure to hold the Fourier series expansion coefficients for  $N=4$ .

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

- [oepdev/libutil/unitary\\_optimizer.h](#)

## 17.28 oepdev::GenEffFrag Class Reference

Generalized Effective Fragment. Container Class.

```
#include <gefp.h>
```

### Public Member Functions

- [GenEffFrag](#) ()  
*Initialize with default name of GEFP (Default)*
- [GenEffFrag](#) (std::string name)  
*Initialize with custom name of GEFP.*
- [~GenEffFrag](#) ()  
*Destruct.*
- void [rotate](#) (std::shared\_ptr< psi::Matrix > R)  
*Rotate.*
- void [translate](#) (std::shared\_ptr< psi::Vector > T)  
*Translate.*
- void [superimpose](#) (std::shared\_ptr< psi::Matrix > targetXYZ, std::vector< int > supList)  
*Superimpose.*
- void [set\\_gefp\\_polarization](#) (const std::shared\_ptr< [GenEffPar](#) > &par)  
*Set the Density Matrix Susceptibility Tensor Object.*
- void [set\\_dmat\\_dipole\\_polarizability](#) (const std::vector< std::vector< std::shared\_ptr< psi::Matrix >>> &susc)  
*Set the Density Matrix Dipole Polarizability.*
- void [set\\_dmat\\_dipole\\_dipole\\_hyperpolarizability](#) (const std::vector< std::vector< std::shared\_ptr< psi::Matrix >>> &susc)  
*Set the Density Matrix Dipole-Dipole Hyperpolarizability.*
- void [set\\_dmat\\_quadrupole\\_polarizability](#) (const std::vector< std::vector< std::shared\_ptr< psi::Matrix >>> &susc)  
*Set the Density Matrix Quadrupole Polarizability.*
- std::shared\_ptr< psi::Matrix > [susceptibility](#) (int fieldRank, int fieldGradientRank, int i, int x) const  
*Grab the Density Matrix Susceptibility.*
- std::vector< std::shared\_ptr< psi::Matrix >> [susceptibility](#) (int fieldRank, int fieldGradientRank, int i) const  
*Grab the Density Matrix Susceptibility.*
- std::vector< std::vector< std::shared\_ptr< psi::Matrix >>> [susceptibility](#) (int fieldRank, int fieldGradientRank) const  
*Grab the Density Matrix Susceptibility.*

### Public Attributes

- std::map< std::string, std::shared\_ptr< [GenEffPar](#) > > [parameters](#)  
*Dictionary of All GEF Parameters.*

## Protected Attributes

- `std::string name_`  
*Name of GEFP.*
- `std::shared_ptr< GenEffPar > densityMatrixSusceptibilityGEF_`  
*Density Matrix Susceptibility Tensor.*
- `std::shared_ptr< GenEffPar > electrostaticEnergyGEF_`  
*Electrostatic Energy Effective One-Electron Potential.*
- `std::shared_ptr< GenEffPar > repulsionEnergyGEF_`  
*Exchange-Repulsion Effective One-Electron Potential.*
- `std::shared_ptr< GenEffPar > chargeTransferEnergyGEF_`  
*Charge-Transfer Effective One-Electron Potential.*
- `std::shared_ptr< GenEffPar > EETCouplingConstantGEF_`  
*EET Coupling Effective One-Electron Potential.*

### 17.28.1 Detailed Description

Generalized Effective Fragment. Container Class.

Describes the GEFP fragment that is in principle designed to work at correlated levels of theory.

### 17.28.2 Member Function Documentation

**17.28.2.1** `std::shared_ptr<psi::Matrix> oepdev::GenEffFrag::susceptibility ( int fieldRank, int fieldGradientRank, int i, int x ) const [inline]`

Grab the Density Matrix Susceptibility.

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field
<i>fieldGradient-Rank</i>	- power dependency with respect to the electric field gradient
<i>i</i>	- id of the distributed site
<i>x</i>	- id of the composite Cartesian component

**17.28.2.2** `std::vector<std::shared_ptr<psi::Matrix> > oepdev::GenEffFrag::susceptibility ( int fieldRank, int fieldGradientRank, int i ) const [inline]`

Grab the Density Matrix Susceptibility.

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field
<i>fieldGradient-Rank</i>	- power dependency with respect to the electric field gradient
<i>i</i>	- id of the distributed site

**17.28.2.3** `std::vector<std::vector<std::shared_ptr<psi::Matrix> > > oepdev::GenEffFrag::susceptibility ( int fieldRank, int fieldGradientRank ) const [inline]`

Grab the Density Matrix Susceptibility.

## Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field
<i>fieldGradient-Rank</i>	- power dependency with respect to the electric field gradient

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

- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp.cc

## 17.29 oepdev::GenEffPar Class Reference

Generalized Effective Fragment Parameters. Container Class.

```
#include <gefp.h>
```

### Public Member Functions

- [GenEffPar](#) (std::string name)  
*Create with name of this parameter type.*
- [~GenEffPar](#) ()  
*Destruct.*
- void [set\\_susceptibility](#) (int fieldRank, int fieldGradientRank, const std::vector< std::vector< std::shared\_ptr< psi::Matrix >>> &susc)  
*Set the Density Matrix Susceptibility.*
- void [set\\_dipole\\_polarizability](#) (const std::vector< std::vector< std::shared\_ptr< psi::Matrix >>> &susc)  
*Set The Density Matrix Dipole Polarizability.*
- void [set\\_dipole\\_dipole\\_hyperpolarizability](#) (const std::vector< std::vector< std::shared\_ptr< psi::Matrix >>> &susc)  
*Set The Density Matrix Dipole-Dipole Hyperpolarizability.*
- void [set\\_quadrupole\\_polarizability](#) (const std::vector< std::vector< std::shared\_ptr< psi::Matrix >>> &susc)  
*Set The Density Matrix Quadrupole Polarizability.*
- void [set\\_centres](#) (const std::vector< std::shared\_ptr< psi::Vector >> &centres)  
*Set the distributed centres' positions.*
- void [allocate](#) (int fieldRank, int fieldGradientRank, int nsites, int nbf)  
*Allocate the Density Matrix Susceptibility.*
- void [allocate\\_dipole\\_polarizability](#) (int nsites, int nbf)  
*Allocate The Density Matrix Dipole Polarizability.*
- void [allocate\\_dipole\\_dipole\\_hyperpolarizability](#) (int nsites, int nbf)  
*Allocate The Density Matrix Dipole-Dipole Hyperpolarizability.*
- void [allocate\\_quadrupole\\_polarizability](#) (int nsites, int nbf)  
*Allocate The Density Matrix Quadrupole Polarizability.*
- bool [hasDensityMatrixDipolePolarizability](#) () const
- bool [hasDensityMatrixDipoleDipoleHyperpolarizability](#) () const
- bool [hasDensityMatrixQuadrupolePolarizability](#) () const
- std::shared\_ptr< psi::Matrix > [susceptibility](#) (int fieldRank, int fieldGradientRank, int i, int x) const  
*Grab the Density Matrix Susceptibility.*
- std::vector< std::shared\_ptr< psi::Matrix >> [susceptibility](#) (int fieldRank, int fieldGradientRank, int i) const  
*Grab the Density Matrix Susceptibility.*

- `std::vector< std::vector< std::shared_ptr< psi::Matrix > > > susceptibility (int fieldRank, int fieldGradientRank) const`  
*Grab the Density Matrix Susceptibility.*
- `std::vector< std::vector< std::shared_ptr< psi::Matrix > > > dipole_polarizability () const`  
*Grab the density matrix dipole polarizability tensor.*
- `std::vector< std::shared_ptr< psi::Matrix > > dipole_polarizability (int i) const`  
*Grab the density matrix dipole polarizability tensor's x-th component.*
- `std::shared_ptr< psi::Matrix > dipole_polarizability (int i, int x) const`  
*Grab the density matrix dipole polarizability tensor's x-th component of the i-th distributed site.*
- `std::vector< std::vector< std::shared_ptr< psi::Matrix > > > dipole_dipole_hyperpolarizability () const`  
*Grab the density matrix dipole-dipole hyperpolarizability tensor.*
- `std::vector< std::shared_ptr< psi::Matrix > > dipole_dipole_hyperpolarizability (int i) const`  
*Grab the density matrix dipole-dipole hyperpolarizability tensor's x-th component.*
- `std::shared_ptr< psi::Matrix > dipole_dipole_hyperpolarizability (int i, int x) const`  
*Grab the density matrix dipole-dipole hyperpolarizability tensor's x-th component of the i-th distributed site.*
- `std::vector< std::vector< std::shared_ptr< psi::Matrix > > > quadrupole_polarizability () const`  
*Grab the density matrix quadrupole polarizability tensor.*
- `std::vector< std::shared_ptr< psi::Matrix > > quadrupole_polarizability (int i) const`  
*Grab the density matrix quadrupole polarizability tensor's x-th component.*
- `std::shared_ptr< psi::Matrix > quadrupole_polarizability (int i, int x) const`  
*Grab the density matrix quadrupole polarizability tensor's x-th component of the i-th distributed site.*
- `std::vector< std::shared_ptr< psi::Vector > > centres () const`  
*Grab the centres' positions.*
- `std::shared_ptr< psi::Vector > centre (int i) const`  
*Grab the position of the i-th distributed site.*
- `std::shared_ptr< psi::Matrix > compute_density_matrix (std::shared_ptr< psi::Vector > field)`  
*Compute the density matrix due to the uniform electric field perturbation.*
- `std::shared_ptr< psi::Matrix > compute_density_matrix (double fx, double fy, double fz)`  
*Compute the density matrix due to the uniform electric field perturbation.*
- `std::shared_ptr< psi::Matrix > compute_density_matrix (std::vector< std::shared_ptr< psi::Vector > > fields)`  
*Compute the density matrix due to the non-uniform electric field perturbation.*
- `std::shared_ptr< psi::Matrix > compute_density_matrix (std::vector< std::shared_ptr< psi::Vector > > fields, std::vector< std::shared_ptr< psi::Matrix > > grads)`  
*Compute the density matrix due to the non-uniform electric field perturbation.*

## Protected Attributes

- `std::string name_`  
*The Name of Parameter Type.*
- `std::vector< std::vector< std::shared_ptr< psi::Matrix > > > densityMatrixDipolePolarizability_`  
*The Density Matrix Dipole Polarizability.*
- `std::vector< std::vector< std::shared_ptr< psi::Matrix > > > densityMatrixDipoleDipoleHyperpolarizability_`

*The Density Matrix Dipole-Dipole Hyperpolarizability.*

- `std::vector< std::vector  
< std::shared_ptr< psi::Matrix > > >` [densityMatrixQuadrupolePolarizability\\_](#)

*The Density Matrix Quadrupole Polarizability.*

- `std::vector< std::shared_ptr  
< psi::Vector > >` [distributedCentres\\_](#)

*The Positions of the Distributed Centres.*

- `bool` [hasDensityMatrixDipolePolarizability\\_](#)
- `bool` [hasDensityMatrixDipoleDipoleHyperpolarizability\\_](#)
- `bool` [hasDensityMatrixQuadrupolePolarizability\\_](#)

### 17.29.1 Detailed Description

Generalized Effective Fragment Parameters. Container Class.

### 17.29.2 Member Function Documentation

**17.29.2.1** `void oepdev::GenEffPar::allocate ( int fieldRank, int fieldGradientRank, int nsites, int nbf )` [\[inline\]](#)

Allocate the Density Matrix Susceptibility.

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field $\mathbf{F}$
<i>fieldGradientRank</i>	- power dependency with respect to the electric field gradient $\nabla \otimes \mathbf{F}$
<i>nsites</i>	- number of distributed sites
<i>nbf</i>	- number of basis functions in the basis set

The following susceptibilities are supported (*fieldRank*, *fieldGradientRank*):

- (1, 0) - dipole polarizability, interacts with  $\mathbf{F}$
- (2, 0) - dipole-dipole hyperpolarizability, interacts with  $\mathbf{F} \otimes \mathbf{F}$
- (0, 1) - quadrupole polarizability, interacts with  $\nabla \otimes \mathbf{F}$

**17.29.2.2** `std::shared_ptr< psi::Matrix > oepdev::GenEffPar::compute_density_matrix ( std::shared_ptr< psi::Vector > field )`

Compute the density matrix due to the uniform electric field perturbation.

Parameters

<i>field</i>	- the uniform electric field vector (A.U.)
--------------	--

**17.29.2.3** `std::shared_ptr< psi::Matrix > oepdev::GenEffPar::compute_density_matrix ( double fx, double fy, double fz )`

Compute the density matrix due to the uniform electric field perturbation.

Parameters

<i>fx</i>	- x-th Cartesian component of the uniform electric field vector (A.U.)
-----------	--

<i>fy</i>	- y-th Cartesian component of the uniform electric field vector (A.U.)
<i>fz</i>	- z-th Cartesian component of the uniform electric field vector (A.U.)

17.29.2.4 `std::shared_ptr< psi::Matrix > oepdev::GenEffPar::compute_density_matrix ( std::vector< std::shared_ptr< psi::Vector >> fields )`

Compute the density matrix due to the non-uniform electric field perturbation.

Parameters

<i>fields</i>	- the list of non-uniform electric field vector (A.U.) evaluated at the distributed DMatPol sites
---------------	---

17.29.2.5 `std::shared_ptr< psi::Matrix > oepdev::GenEffPar::compute_density_matrix ( std::vector< std::shared_ptr< psi::Vector >> fields, std::vector< std::shared_ptr< psi::Matrix >> grads )`

Compute the density matrix due to the non-uniform electric field perturbation.

Parameters

<i>fields</i>	- the list of electric field vectors (A.U.) evaluated at the distributed DMatPol sites
<i>grads</i>	- the list of electric field gradient matrices (A.U.) evaluated at the distributed DMatPol sites

17.29.2.6 `void oepdev::GenEffPar::set_susceptibility ( int fieldRank, int fieldGradientRank, const std::vector< std::vector< std::shared_ptr< psi::Matrix >>> & susc ) [inline]`

Set the Density Matrix Susceptibility.

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field $\mathbf{F}$
<i>fieldGradient-Rank</i>	- power dependency with respect to the electric field gradient $\nabla \otimes \mathbf{F}$
<i>susc</i>	- the susceptibility tensor

The following susceptibilities are supported (fieldRank, fieldGradientRank):

- (1, 0) - dipole polarizability, interacts with  $\mathbf{F}$
- (2, 0) - dipole-dipole hyperpolarizability, interacts with  $\mathbf{F} \otimes \mathbf{F}$
- (0, 1) - quadrupole polarizability, interacts with  $\nabla \otimes \mathbf{F}$

17.29.2.7 `std::shared_ptr< psi::Matrix > oepdev::GenEffPar::susceptibility ( int fieldRank, int fieldGradientRank, int i, int x ) const [inline]`

Grab the Density Matrix Susceptibility.

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field
<i>fieldGradient-Rank</i>	- power dependency with respect to the electric field gradient



$i$	- id of the distributed site
$x$	- id of the composite Cartesian component

The following susceptibilities are supported (fieldRank, fieldGradientRank):

- (1, 0) - dipole polarizability, interacts with  $\mathbf{F}$
- (2, 0) - dipole-dipole hyperpolarizability, interacts with  $\mathbf{F} \otimes \mathbf{F}$
- (0, 1) - quadrupole polarizability, interacts with  $\nabla \otimes \mathbf{F}$

The distributed sites are assumed to be atomic sites or molecular orbital centroids (depending on the polarization factory used). For the electric field, the composite Cartesian index is just an ordinary Cartesian index. For the electric field gradient and electric field squared, the composite Cartesian index is given as

$$I(x, y) = 3x + y$$

where the values of 0, 1 and 2 correspond to  $x$ ,  $y$  and  $z$  Cartesian components, respectively. Therefore, in the latter case, there is 9 distinct composite Cartesian components.

**17.29.2.8** `std::vector<std::shared_ptr<psi::Matrix> > oepdev::GenEffPar::susceptibility ( int fieldRank, int fieldGradientRank, int i ) const [inline]`

Grab the Density Matrix Susceptibility.

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field
<i>fieldGradientRank</i>	- power dependency with respect to the electric field gradient
<i>i</i>	- id of the distributed site

The following susceptibilities are supported (fieldRank, fieldGradientRank):

- (1, 0) - dipole polarizability, interacts with  $\mathbf{F}$
- (2, 0) - dipole-dipole hyperpolarizability, interacts with  $\mathbf{F} \otimes \mathbf{F}$
- (0, 1) - quadrupole polarizability, interacts with  $\nabla \otimes \mathbf{F}$

The distributed sites are assumed to be atomic sites or molecular orbital centroids (depending on the polarization factory used).

**17.29.2.9** `std::vector<std::vector<std::shared_ptr<psi::Matrix> > > oepdev::GenEffPar::susceptibility ( int fieldRank, int fieldGradientRank ) const [inline]`

Grab the Density Matrix Susceptibility.

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field
<i>fieldGradientRank</i>	- power dependency with respect to the electric field gradient

The following susceptibilities are supported (fieldRank, fieldGradientRank):

- (1, 0) - dipole polarizability, interacts with  $\mathbf{F}$
- (2, 0) - dipole-dipole hyperpolarizability, interacts with  $\mathbf{F} \otimes \mathbf{F}$
- (0, 1) - quadrupole polarizability, interacts with  $\nabla \otimes \mathbf{F}$

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

- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp.cc



- psi::Options & [options\\_](#)  
*Psi4 Options.*
- std::default\_random\_engine [randomNumberGenerator\\_](#)  
*Random number generators.*
- std::uniform\_real\_distribution  
< double > [randomDistribution\\_](#)
- std::shared\_ptr< psi::Matrix > [excludeSpheres\\_](#)  
*Matrix with vdW sphere information.*
- std::map< std::string, double > [vdwRadius\\_](#)  
*Map with vdW radii.*
- double [cx\\_](#)  
*Centre-of-mass coordinates.*
- double [cy\\_](#)
- double [cz\\_](#)
- double [radius\\_](#)  
*Radius of padding sphere around the molecule.*
- const int [nbf\\_](#)  
*Number of basis functions.*
- std::shared\_ptr< CPHF > [cphfSolver\\_](#)  
*The CPHF object.*
- std::shared\_ptr  
< [oepdev::GenEffParFactory](#) > [abInitioPolarizationSusceptibilitiesFactory\\_](#)  
*Ab initio polarization susceptibility factory.*

### 17.30.1 Detailed Description

Generalized Effective Fragment Factory. Abstract Base.

Describes the GEFP fragment that is in principle designed to work at correlated levels of theory.

### 17.30.2 Member Function Documentation

**17.30.2.1** `std::shared_ptr< oepdev::GenEffParFactory > oepdev::GenEffParFactory::build ( const std::string & type, std::shared_ptr< psi::Wavefunction > wfn, psi::Options & opt ) [static]`

Build Density Matrix Susceptibility Generalized Factory.

Parameters

<i>type</i>	- Type of factory
<i>wfn</i>	- Psi4 wavefunction
<i>opt</i>	- Psi4 options

Available factory types:

- POLARIZATION - creates the polarization generalized effective fragment parameters' factory Factory sub-type is specified in Psi4 options (input file).

#### Note

Useful options:

- POLARIZATION factory type:
  - DMATPOL\_TRAINING\_MODE - training mode. Default: EFIELD
  - DMATPOL\_NSAMPLES - number of random samples (field or test charges sets). Default: 30

- DMATPOL\_FIELD\_SCALE - electric field scale factor (relevant if training mode is EFIELD). Default: 0.01 [au]
- DMATPOL\_NTEST\_CHARGE - number of test charges per sample (relevant if training mode is CHARGES). Default: 1
- DMATPOL\_TEST\_CHARGE - test charge value (relevant if training mode is CHARGES). Default: 0.001 [au]
- DMATPOL\_FIELD\_RANK - electric field rank. Default: 1
- DMATPOL\_GRADIENT\_RANK - electric field gradient rank. Default: 0
- DMATPOL\_TEST\_FIELD\_X - test electric field in X direction. Default: 0.000 [au]
- DMATPOL\_TEST\_FIELD\_Y - test electric field in Y direction. Default: 0.000 [au]
- DMATPOL\_TEST\_FIELD\_Z - test electric field in Z direction. Default: 0.008 [au]
- DMATPOL\_OUT\_STATS - output file name for statistical evaluation results. Default: dmatpol.-stats.dat
- DMATPOL\_DO\_AB\_INITIO - compute ab initio susceptibilities and evaluate statistics for it. Default: false
- DMATPOL\_OUT\_STATS\_AB\_INITIO - output file name for statistical evaluation results of ab initio model. Default: dmatpol.stats.abinitio.dat

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

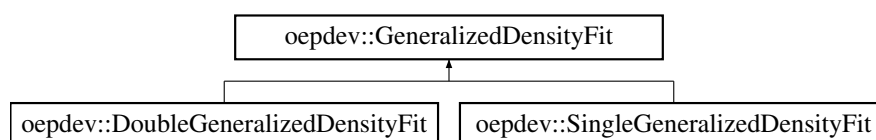
- [oepdev/libgefp/gefp.h](#)
- [oepdev/libgefp/gefp.cc](#)

## 17.31 oepdev::GeneralizedDensityFit Class Reference

Generalized Density Fitting Scheme. Abstract Base.

```
#include <oep_gdf.h>
```

Inheritance diagram for oepdev::GeneralizedDensityFit:



### Public Member Functions

- [GeneralizedDensityFit](#) ()  
*Constructor. Initializes the pointers.*
- virtual [~GeneralizedDensityFit](#) ()  
*Destructor.*
- virtual std::shared\_ptr< psi::Matrix > [compute](#) (void)=0  
*Perform the generalized density fit.*
- std::shared\_ptr< psi::Matrix > [G](#) (void) const  
*Extract the  $G_{\xi_i}$  coefficients.*

## Static Public Member Functions

- static `std::shared_ptr`  
`< GeneralizedDensityFit > build` (`std::shared_ptr< psi::BasisSet > bs_auxiliary`, `std::shared_ptr< psi::Matrix > v_vector`)  
*Factory for Single GDF Computer.*
- static `std::shared_ptr`  
`< GeneralizedDensityFit > build` (`std::shared_ptr< psi::BasisSet > bs_auxiliary`, `std::shared_ptr< psi::BasisSet > bs_intermediate`, `std::shared_ptr< psi::Matrix > v_vector`)  
*Factory for Double GDF Computer.*

## Protected Member Functions

- void `invert_matrix` (`std::shared_ptr< psi::Matrix > &M`)  
*Invert a square matrix and check if the inverse is acceptable.*

## Protected Attributes

- `std::shared_ptr< psi::Matrix > G_`  
*The OEP coefficients  $G_{\xi i}$ .*
- `std::shared_ptr< psi::Matrix > H_`  
*The intermediate DF coefficients for  $\hat{v}|i\rangle$ .*
- `std::shared_ptr< psi::Matrix > V_`  
*The V matrix ( $\xi|\hat{v}i\rangle$ ).*
- int `n_a_`  
*Number of auxiliary basis set functions.*
- int `n_i_`  
*Number of intermediate basis set functions.*
- int `n_o_`  
*Number of OEP's.*
- `std::shared_ptr< psi::BasisSet > bs_a_`  
*Basis set: auxiliary.*
- `std::shared_ptr< psi::BasisSet > bs_i_`  
*Basis set: intermediate.*
- `std::shared_ptr`  
`< oepdev::IntegralFactory > ints_aa_`  
*Integral factory: aux - aux.*
- `std::shared_ptr`  
`< oepdev::IntegralFactory > ints_ai_`  
*Integral factory: aux - int.*
- `std::shared_ptr`  
`< oepdev::IntegralFactory > ints_ii_`  
*Integral factory: int - int.*

### 17.31.1 Detailed Description

Generalized Density Fitting Scheme. Abstract Base.

Performs the following map:

$$\hat{v}|i\rangle \cong \sum_{\eta} G_{\eta i} |\eta\rangle$$

where  $\hat{v}$  is the effective one-electron potential (OEP) operator,  $|i\rangle$  is an arbitrary state vector and  $|\eta\rangle$  is an auxiliary basis vector. The coefficients  $G_{\eta i}$  are stored and define the OEP acting on the state  $i$ . The mapping onto the auxiliary space can be done in two ways:

- **Single Density Fit.** [This method](#) requires the auxiliary basis set to be nearly complete.
- **Double Density Fit.** [This method](#) can be used to arbitrary auxiliary basis sets.

### 17.31.2 Member Function Documentation

17.31.2.1 `std::shared_ptr< GeneralizedDensityFit > GeneralizedDensityFit::build ( std::shared_ptr< psi::BasisSet > bs_auxiliary, std::shared_ptr< psi::Matrix > v_vector ) [static]`

Factory for Single GDF Computer.

Parameters

<i>bs_auxiliary</i>	- auxiliary basis set
<i>v_vector</i>	- the matrix with $V_{\xi i}$ elements

Returns

Generalized Density Fit Computer.

17.31.2.2 `std::shared_ptr< GeneralizedDensityFit > GeneralizedDensityFit::build ( std::shared_ptr< psi::BasisSet > bs_auxiliary, std::shared_ptr< psi::BasisSet > bs_intermediate, std::shared_ptr< psi::Matrix > v_vector ) [static]`

Factory for Double GDF Computer.

Parameters

<i>bs_auxiliary</i>	- auxiliary basis set
<i>bs_intermediate</i>	- intermediate basis set
<i>v_vector</i>	- the matrix with $V_{\xi i}$ elements

Returns

Generalized Density Fit Computer.

17.31.2.3 `std::shared_ptr< psi::Matrix > GeneralizedDensityFit::compute ( void ) [pure virtual]`

Perform the generalized density fit.

Returns

The OEP coefficients  $G_{\xi i}$

Implemented in [oepdev::DoubleGeneralizedDensityFit](#), and [oepdev::SingleGeneralizedDensityFit](#).

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

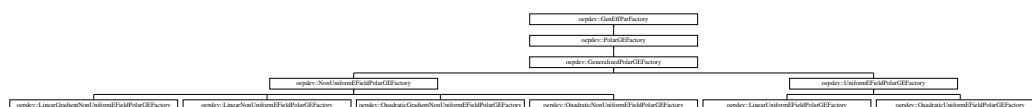
- [oepdev/liboep/oep\\_gdf.h](#)
- [oepdev/liboep/oep\\_gdf.cc](#)

## 17.32 oepdev::GeneralizedPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::GeneralizedPolarGEFactory:



## Classes

- struct [StatisticalSet](#)  
A structure to handle statistical data.

## Public Member Functions

- [GeneralizedPolarGEFactory](#) (std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)  
Construct from Psi4 wavefunction and options.
- virtual [~GeneralizedPolarGEFactory](#) ()  
Destruct.
- virtual std::shared\_ptr< [GenEffPar](#) > [compute](#) (void)  
Perform Least-Squares Fit.
- bool [has\\_dipole\\_polarizability](#) () const  
Dipole Polarizability (interacting with  $\mathbf{F}$ )
- bool [has\\_dipole\\_dipole\\_hyperpolarizability](#) () const  
Dipole-Dipole Hyperpolarizability (interacting with  $\mathbf{F}^2$ )
- bool [has\\_quadrupole\\_polarizability](#) () const  
Quadrupole Polarizability (interacting with  $\nabla \otimes \mathbf{F}$ )
- bool [has\\_ab\\_initio\\_dipole\\_polarizability](#) () const  
Ab Initio Dipole Polarizability (interacting with  $\mathbf{F}$ )
- double [Zinit](#) () const  
Grab initial summaric Z value.
- double [Z](#) () const  
Grab final summaric Z value.

## Protected Member Functions

- void [allocate](#) (void)  
Allocate memory.
- void [invert\\_hessian](#) (void)  
Invert Hessian (do also the identity test)
- void [compute\\_electric\\_field\\_sums](#) (void)  
Compute electric field sum set.
- void [compute\\_electric\\_field\\_gradient\\_sums](#) (void)  
Compute electric field gradient sum set.
- void [compute\\_statistics](#) (void)  
Run the statistical evaluation of results.
- void [set\\_distributed\\_centres](#) (void)  
Set the distributed centres.
- void [compute\\_parameters](#) (void)  
Compute the parameters.
- void [fit](#) (void)

- *Perform least-squares fit.*
- void [compute\\_ab\\_initio](#) (void)  
*Compute ab initio parameters.*
- void [save](#) (int i, int j)  
*Save susceptibility tensors associated with the i-th and j-th basis set function.*
- virtual void [compute\\_samples](#) (void)=0  
*Compute samples of density matrices and select electric field distributions.*
- virtual void [compute\\_gradient](#) (int i, int j)=0  
*Compute Gradient vector associated with the i-th and j-th basis set function.*
- virtual void [compute\\_hessian](#) (void)=0  
*Compute Hessian matrix (independent on the parameters)*

## Protected Attributes

- int [nBlocks\\_](#)  
*Number of parameter blocks.*
- int [nSites\\_](#)  
*Number of distributed sites.*
- int [nSitesAbInitio\\_](#)  
*Number of distributed sites of Ab Initio model (FF - single site (com); distributed: LMO sites)*
- int [nParameters\\_](#)  
*Dimensionality of entire parameter space.*
- std::vector< int > [nParametersBlock\\_](#)  
*Dimensionality of parameter space per block.*
- const int [nSamples\\_](#)  
*Number of statistical samples.*
- const double [symmetryNumber\\_](#) [6]  
*Symmetry number for matrix susceptibilities.*
- std::shared\_ptr< psi::Matrix > [Gradient\\_](#)  
*Gradient.*
- std::shared\_ptr< psi::Matrix > [Hessian\\_](#)  
*Hessian.*
- std::shared\_ptr< psi::Matrix > [Parameters\\_](#)  
*Parameters.*
- std::shared\_ptr  
< oepdev::GenEffPar > [PolarizationSusceptibilities\\_](#)  
*Density Matrix Susceptibility Tensors Object.*
- std::shared\_ptr  
< oepdev::GenEffPar > [abInitioPolarizationSusceptibilities\\_](#)  
*Density Matrix Susceptibility Tensors Object for Ab Initio Model.*
- bool [hasDipolePolarizability\\_](#)  
*Has Dipole Polarizability?*
- bool [hasDipoleDipoleHyperpolarizability\\_](#)  
*Has Dipole-Dipole Hyperpolarizability?*
- bool [hasQuadrupolePolarizability\\_](#)  
*Has Quadrupole Polarizability?*
- bool [hasAbInitioDipolePolarizability\\_](#)  
*Has Ab Initio Dipole Polarizability?*
- [StatisticalSet](#) [referenceStatisticalSet\\_](#)  
*Reference statistical data.*
- [StatisticalSet](#) [referenceDpolStatisticalSet\\_](#)



- *Multipole reference statistical data.*  
[StatisticalSet modelStatisticalSet\\_](#)
- *Model statistical data.*  
[StatisticalSet abInitioModelStatisticalSet\\_](#)
- *Ab Initio Model statistical data.*  
 • `std::vector< std::shared_ptr< psi::Matrix > >` [VMatrixSet\\_](#)  
*Potential matrix set.*
- `std::vector< std::vector< std::shared_ptr< Vector > > >` [electricFieldSet\\_](#)  
*Electric field set.*
- `std::vector< std::vector< std::shared_ptr< Matrix > > >` [electricFieldGradientSet\\_](#)  
*Electric field gradient set.*
- `std::vector< std::vector< double > >` [electricFieldSumSet\\_](#)  
*Electric field sum set.*
- `std::vector< std::vector< std::shared_ptr< psi::Vector > > >` [electricFieldGradientSumSet\\_](#)  
*Electric field gradient sum set.*
- `std::vector< std::vector< std::shared_ptr< Vector > > >` [abInitioModelElectricFieldSet\\_](#)  
*Electric field set for Ab Initio Model (LMO-distributed)*
- `const double` [mField\\_](#)  
*Level shifters for Hessian blocks.*
- `double` [Zinit\\_](#)  
*Initial summaric Z value.*
- `double` [Z\\_](#)  
*Final summaric Z value.*
- `std::shared_ptr< psi::JK >` [jk\\_](#)  
*Computer of generalized JK objects.*

## Additional Inherited Members

### 17.32.1 Detailed Description

Polarization GEFP Factory with Least-Squares Parameterization.

Implements a general class of methods for the density matrix susceptibility tensors represented by:

$$\delta D_{\alpha\beta} = \sum_i \left\{ \mathbf{B}_{i;\alpha\beta}^{(10)} \cdot \mathbf{F}(\mathbf{r}_i) + \mathbf{B}_{i;\alpha\beta}^{(20)} : \mathbf{F}(\mathbf{r}_i) \otimes \mathbf{F}(\mathbf{r}_i) + \mathbf{B}_{i;\alpha\beta}^{(01)} : \nabla_i \otimes \mathbf{F}(\mathbf{r}_i) + \dots \right\}$$

where:

- $\mathbf{B}_{i;\alpha\beta}^{(10)}$  is the density matrix dipole polarizability
- $\mathbf{B}_{i;\alpha\beta}^{(20)}$  is the density matrix dipole-dipole hyperpolarizability
- $\mathbf{B}_{i;\alpha\beta}^{(01)}$  is the density matrix quadrupole polarizability

all defined for the generalized distributed site at  $\mathbf{r}_i$ .

Available models:

## 1. Training against uniform electric fields

- [oepdev::LinearUniformEFieldPolarGEFactory](#) - linear with respect to electric field
- [oepdev::QuadraticUniformEFieldPolarGEFactory](#) - quadratic with respect to electric field

## 2. Training against non-uniform electric fields

- [oepdev::LinearNonUniformEFieldPolarGEFactory](#) - linear with respect to electric field, distributed site model
- [oepdev::QuadraticNonUniformEFieldPolarGEFactory](#) - quadratic with respect to electric field, distributed site model
- [oepdev::LinearGradientNonUniformEFieldPolarGEFactory](#) - linear with respect to electric field and linear with respect to electric field gradient, distributed site model. This model does not function now.
- [oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory](#) - linear with respect to electric field and linear with respect to electric field gradient, distributed site model. This model does not function now.

For the non-linear field training, a set of point charges in each training sample is assumed. Distributed models use atomic centers as expansion points.

**Determination of the generalized susceptibilities**

Let  $\{\mathbf{F}^{(1)}(\mathbf{r}), \mathbf{F}^{(2)}(\mathbf{r}), \dots, \mathbf{F}^{(N)}(\mathbf{r}), \dots\}$  be a set of  $N_{\max}$  distinct and randomly sampled spatial distributions of electric field. It is assumed that the exact difference one-particle density matrices (with respect to the unperturbed state) defined as

$$\delta\overline{\mathbf{D}}^{(N)} \equiv \overline{\mathbf{D}}^{(N)} - \overline{\mathbf{D}}^{(0)}$$

are known for each sample (overline symbolizes the exact estimate). Now, for each pair of the AO indices the following parameterization is constructed:

$$\delta D^{(N)} = \sum_i^M \left\{ \sum_u^{x,y,z} s_{iu}^{[1]} F_{iu}^{(N)} + \sum_u^{x,y,z} \sum_{w < u} r_{uw} s_{iww}^{[2]} F_{iu}^{(N)} F_{iw}^{(N)} + \dots \right\}$$

(the Greek subscripts were omitted here for notational simplicity). In the above equation,  $B_u^{(i;1)} = s_{iu}^{[1]}$  and  $B_{uw}^{(i;2)} = r_{uw} s_{iww}^{[2]}$ , where  $r_{uw}$  is the symmetry factor equal to 1 for diagonal elements and 2 for off-diagonal elements of  $B_{uw}^{(i;2)}$ . The multiple parameter blocks ( $s^{[1]}$ ,  $s^{[2]}$  and so on) appear in the first power, allowing for linear least-squares regression. The square bracket superscripts denote the block of the parameter space.

To determine the optimum set,  $\mathbf{s} = (s^{[1]} \quad s^{[2]} \quad \dots)^T$ , a loss function  $Z$  that is subject to the least-squares minimization, is defined as

$$Z(\mathbf{s}) = \sum_N^{N_{\max}} \left( \delta D^{(N)} - \delta\overline{\mathbf{D}}^{(N)} \right)^2.$$

The Hessian of  $Z$  computed with respect to the parameters is parameter-independent (constant) and generally non-singular as long as the electric fields on all distributed sites are different. Therefore, the exact solution for the optimal parameters is given by the Newton equation

$$\mathbf{s} = -\mathbf{H}^{-1} \cdot \mathbf{g},$$

where  $\mathbf{g}$  and  $\mathbf{H}$  are the gradient vector and the Hessian matrix, respectively. Note that in this case the dimensions of parameter space for the block 1 and 2 are equal to  $3M$  and  $6M$ , respectively. The explicit forms of the gradient and Hessian up to second-order are given in the next section.

**Explicit Formulae for Gradient and Hessian Blocks in Linear Regression DMS Model**

The gradient vector  $\mathbf{g}$  and the Hessian matrix  $\mathbf{H}$  are built from blocks associated with a particular type of parameters, i.e.,

$$\mathbf{g} = \begin{pmatrix} \mathbf{g}^{[1]} \\ \mathbf{g}^{[2]} \end{pmatrix}, \quad \mathbf{H} = \begin{pmatrix} \mathbf{H}^{[11]} & \mathbf{H}^{[12]} \\ \mathbf{H}^{[21]} & \mathbf{H}^{[22]} \end{pmatrix},$$

where the block indices 1 and 2 correspond to the first- and second-order susceptibilities, respectively. Note that the second derivatives of  $\delta D^{(N)}$  with respect to the adjustable parameters vanish due to the linear functional form of

the parameterization formula given in the previous section. Thus, the gradient element of the  $r$ -th block and Hessian element of the  $(rs)$ -th block read

$$g^{[r]} \equiv \frac{\partial Z}{\partial s^{[r]}} = -2 \sum_N \overline{\delta D}^{(N)} \frac{\partial [\delta D^{(N)}]}{\partial s^{[r]}} ,$$

$$H^{[rs]} \equiv \frac{\partial^2 Z}{\partial s^{[r]} \partial s^{[s]}} = 2 \sum_N \frac{\partial [\delta D^{(N)}]}{\partial s^{[r]}} \frac{\partial [\delta D^{(N)}]}{\partial s^{[s]}} .$$

The explicit formulae for the gradient are

$$g_{ku}^{[1]} = -2 \sum_N \overline{\delta D}^{(N)} F_{ku}^{(N)} ,$$

$$g_{kuw}^{[2]} = -2 r_{uw} \sum_N \overline{\delta D}^{(N)} F_{ku}^{(N)} F_{kw}^{(N)} .$$

The Hessian subsequently follows to be %

$$H_{ku,lw}^{[11]} = 2 \sum_N F_{ku}^{(N)} F_{lw}^{(N)} ,$$

$$H_{ku,lu'w'}^{[12]} = 2 r_{u'w'} \sum_N F_{ku}^{(N)} F_{lu'}^{(N)} F_{lw'}^{(N)} ,$$

$$H_{kuw,lu'w'}^{[22]} = 2 r_{uw} r_{u'w'} \sum_N F_{ku}^{(N)} F_{kw}^{(N)} F_{lu'}^{(N)} F_{lw'}^{(N)} .$$

Note that due to the symmetry of the Hessian matrix, the block 21 is a transpose of the block 12. The composite indices  $ku$  and  $kuw$  are constructed from the distributed site index  $k$  and the appropriate symmetry-adapted ( $w < u$ ) Cartesian component of a particular DMS tensor:  $u$  for the first-order, and  $uw$  for the second-order susceptibility tensor, respectively. The method described above can be easily extended to third and higher orders.

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

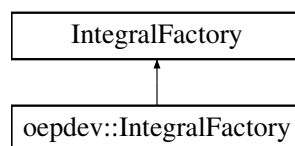
- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp\_polar\_base.cc

## 17.33 oepdev::IntegralFactory Class Reference

Extended [IntegralFactory](#) for computing integrals.

```
#include <integral.h>
```

Inheritance diagram for oepdev::IntegralFactory:



### Public Member Functions

- [IntegralFactory](#) (std::shared\_ptr< psi::BasisSet > bs1, std::shared\_ptr< psi::BasisSet > bs2, std::shared\_ptr< psi::BasisSet > bs3, std::shared\_ptr< psi::BasisSet > bs4)  
Initialize integral factory given a BasisSet for each center. Becomes (bs1 bs2 | bs3 bs4).
- [IntegralFactory](#) (std::shared\_ptr< psi::BasisSet > bs1, std::shared\_ptr< psi::BasisSet > bs2)  
Initialize integral factory given a BasisSet for two centres. Becomes (bs1 bs2 | bs1 bs2).

- [IntegralFactory](#) (std::shared\_ptr< psi::BasisSet > bs1)  
*Initialize integral factory given a BasisSet for two centres. Becomes (bs1 bs1 | bs1 bs1).*
- virtual [~IntegralFactory](#) ()  
*Destructor.*
- virtual [oepdev::TwoBodyAOInt \\* eri\\_1\\_1](#) (int deriv=0, bool use\_shell\_pairs=false)  
*Returns an [ERI\\_1\\_1](#) integral object.*
- virtual [oepdev::TwoBodyAOInt \\* eri\\_2\\_1](#) (int deriv=0, bool use\_shell\_pairs=false)  
*Returns an [ERI\\_2\\_1](#) integral object.*
- virtual [oepdev::TwoBodyAOInt \\* eri\\_2\\_2](#) (int deriv=0, bool use\_shell\_pairs=false)  
*Returns an [ERI\\_2\\_2](#) integral object.*
- virtual [oepdev::TwoBodyAOInt \\* eri\\_3\\_1](#) (int deriv=0, bool use\_shell\_pairs=false)  
*Returns an [ERI\\_3\\_1](#) integral object.*

### 17.33.1 Detailed Description

Extended [IntegralFactory](#) for computing integrals.

In addition to integrals available in Psi4, [oepdev::IntegralFactory](#) enables to compute also:

- OEI's:
  - none at that moment
- ERI's:
  - integrals of type (a|b) - [oepdev::ERI\\_1\\_1](#)
  - integrals of type (ab|c) - [oepdev::ERI\\_2\\_1](#)
  - integrals of type (abc|d) - [oepdev::ERI\\_3\\_1](#)
  - integrals of type (ab|cd) - [oepdev::ERI\\_2\\_2](#) (also in Psi4 as `psi::ERI`)

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

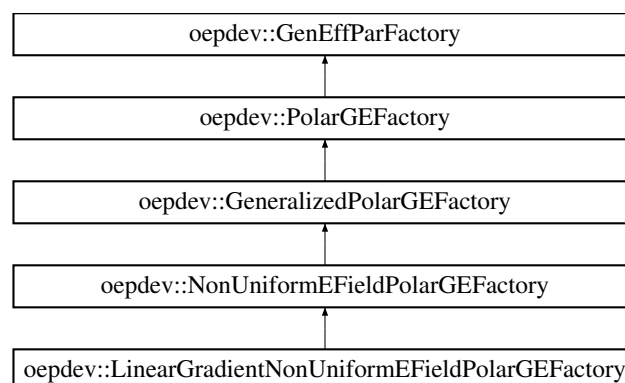
- [oepdev/libpsi/integral.h](#)
- [oepdev/libpsi/integral.cc](#)

## 17.34 oepdev::LinearGradientNonUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for [oepdev::LinearGradientNonUniformEFieldPolarGEFactory](#):



## Public Member Functions

- **LinearGradientNonUniformEFieldPolarGEFactory** (std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- void [compute\\_gradient](#) (int i, int j)  
*Compute Gradient vector associated with the i-th and j-th basis set function.*
- void [compute\\_hessian](#) (void)  
*Compute Hessian matrix (independent on the parameters)*

## Additional Inherited Members

### 17.34.1 Detailed Description

Polarization GEFP Factory with Least-Squares Parameterization.

Implements the generalized density matrix susceptibility model of the form

$$\delta D_{\alpha\beta} \approx \sum_i \left\{ \mathbf{B}_{i;\alpha\beta}^{(10)} \cdot \mathbf{F}(\mathbf{r}_i) + \mathbf{B}_{i;\alpha\beta}^{(01)} : \nabla_i \otimes \mathbf{F}(\mathbf{r}_i) \right\}$$

where:

- $\mathbf{B}_{i;\alpha\beta}^{(10)}$  is the density matrix dipole polarizability
- $\mathbf{B}_{i;\alpha\beta}^{(01)}$  is the density matrix quadrupole polarizability all defined for the distributed site at  $\mathbf{r}_i$ .

#### Note

This model is not available now and probably will be deprecated in the future.

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

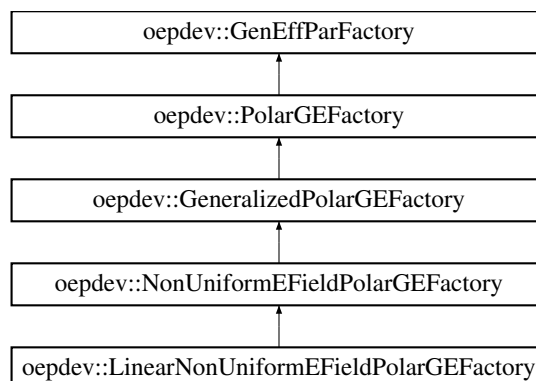
- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp\_polar\_nonuniform\_field\_1\_grad\_1.cc

## 17.35 oepdev::LinearNonUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::LinearNonUniformEFieldPolarGEFactory:



## Public Member Functions

- **LinearNonUniformEFieldPolarGEFactory** (std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- void [compute\\_gradient](#) (int i, int j)  
*Compute Gradient vector associated with the i-th and j-th basis set function.*
- void [compute\\_hessian](#) (void)  
*Compute Hessian matrix (independent on the parameters)*

## Additional Inherited Members

### 17.35.1 Detailed Description

Polarization GEFP Factory with Least-Squares Parameterization.

Implements the generalized density matrix susceptibility model of the form

$$\delta D_{\alpha\beta} \approx \sum_i \mathbf{B}_{i;\alpha\beta}^{(10)} \cdot \mathbf{F}(\mathbf{r}_i)$$

where:

- $\mathbf{B}_{i;\alpha\beta}^{(10)}$  is the density matrix dipole polarizability defined for the distributed site at  $\mathbf{r}_i$ .

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

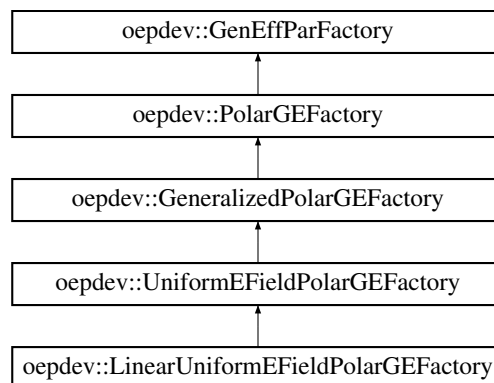
- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp\_polar\_nonuniform\_field\_1.cc

## 17.36 oepdev::LinearUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::LinearUniformEFieldPolarGEFactory:



## Public Member Functions

- **LinearUniformEFieldPolarGEFactory** (std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- void [compute\\_gradient](#) (int i, int j)  
*Compute Gradient vector associated with the i-th and j-th basis set function.*
- void [compute\\_hessian](#) (void)  
*Compute Hessian matrix (independent on the parameters)*

## Additional Inherited Members

### 17.36.1 Detailed Description

Polarization GEFP Factory with Least-Squares Parameterization.

Implements the generalized density matrix susceptibility model of the form

$$\delta D_{\alpha\beta} \approx \mathbf{B}_{\alpha\beta}^{(10)} \cdot \mathbf{F}$$

where:

- $\mathbf{B}_{\alpha\beta}^{(10)}$  is the density matrix dipole polarizability

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

- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp\_polar\_uniform\_field\_1.cc

## 17.37 oepdev::MultipoleConvergence Class Reference

Multipole Convergence.

```
#include <dmtip.h>
```

### Public Types

- enum **ConvergenceLevel** {  
  **R1**, **R2**, **R3**, **R4**,  
  **R5** }
- enum **Property** { **Energy**, **Potential** }

### Public Member Functions

- **MultipoleConvergence** (std::shared\_ptr< [DMTPole](#) > dmtip1, std::shared\_ptr< [DMTPole](#) > dmtip2, ConvergenceLevel max\_clevel=R4)
- void **compute** (Property property=Energy)
- std::shared\_ptr< psi::Vector > **level** (ConvergenceLevel clevel=R4)

### Protected Member Functions

- void **compute\_energy** ()
- void **compute\_potential** ()

### Protected Attributes

- ConvergenceLevel **max\_clevel\_**
- std::shared\_ptr< [DMTPole](#) > **dmtip\_1\_**
- std::shared\_ptr< [DMTPole](#) > **dmtip\_2\_**
- std::map< std::string, std::shared\_ptr< psi::Vector > > **convergenceList\_**

### 17.37.1 Detailed Description

Multipole Convergence.

Handles the convergence of the distributed multipole expansions up to hexadecapole.

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

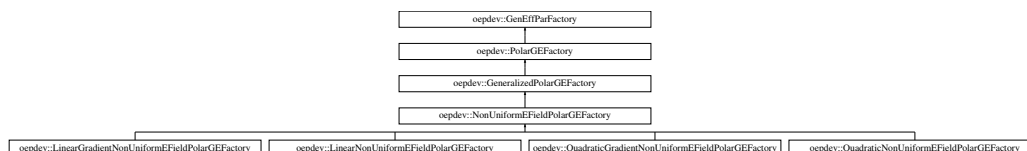
- [oepdev/lib3d/dmtp.h](#)
- [oepdev/lib3d/dmtp\\_base.cc](#)

## 17.38 oepdev::NonUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::NonUniformEFieldPolarGEFactory:



### Public Member Functions

- **NonUniformEFieldPolarGEFactory** (std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- void [compute\\_samples](#) (void)  
*Compute samples of density matrices and select electric field distributions.*
- virtual void [compute\\_gradient](#) (int i, int j)=0  
*Compute Gradient vector associated with the i-th and j-th basis set function.*
- virtual void [compute\\_hessian](#) (void)=0  
*Compute Hessian matrix (independent on the parameters)*

### Additional Inherited Members

#### 17.38.1 Detailed Description

Polarization GEFP Factory with Least-Squares Parameterization.

Implements a class of density matrix susceptibility models for parameterization in the non-uniform electric field generated by point charges.

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

- [oepdev/libgefp/gefp.h](#)
- [oepdev/libgefp/gefp\\_polar\\_nonuniform\\_base.cc](#)

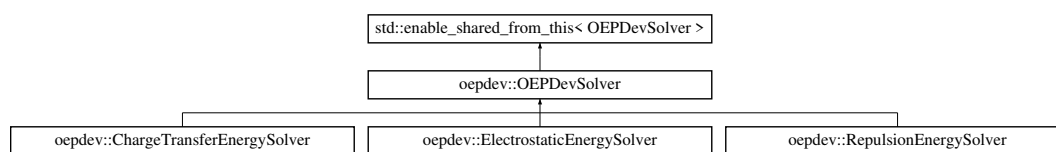
## 17.39 oepdev::OEPDevSolver Class Reference

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\\_oeplibased](#) (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\\_oeplibased\\_](#)  
*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.*

### 17.39.1 Detailed Description

Solver of properties of molecular aggregates. Abstract base.

Uses only a wavefunction union object to initialize. Available solvers:

- ELECTROSTATIC ENERGY
- REPULSION ENERGY
- CHARGE TRANSFER ENERGY

Options controlling the generalized density fitting (GDF):

- OEPDEV\_DF\_TYPE - type of the GDF. Default: DOUBLE.
- DF\_BASIS\_OEP - auxiliary basis set. Default: sto-3g.
- DF\_BASIS\_INT - intermediate basis set. Relevant only if double GDF is used. Default: aug-cc-pVDZ-jkfit. Note that intermediate basis set should be nearly complete.

## 17.39.2 Constructor & Destructor Documentation

### 17.39.2.1 OEPDevSolver::OEPDevSolver ( SharedWavefunctionUnion *wfn\_union* )

Take wavefunction union and initialize the Solver.

Parameters

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

## 17.39.3 Member Function Documentation

### 17.39.3.1 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](#)

### 17.39.3.2 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::ChargeTransferEnergySolver](#), [oepdev::RepulsionEnergySolver](#), and [oepdev::ElectrostaticEnergySolver](#).

### 17.39.3.3 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::ChargeTransferEnergySolver](#), [oepdev::RepulsionEnergySolver](#), and [oepdev::ElectrostaticEnergySolver](#).

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

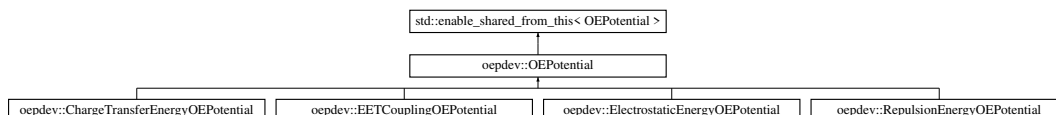
- oepdev/libsolver/solver.h
- oepdev/libsolver/solver\_base.cc
- oepdev/libsolver/solver\_energy\_pauli.cc

## 17.40 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)  
*Fully ESP-based OEP object.*
- **OEPotential** (SharedWavefunction [wfn](#), SharedBasisSet auxiliary, SharedBasisSet intermediate, Options &options)  
*General OEP object.*
- virtual **~OEPotential** ()  
*Destructor.*
- virtual void **compute** (void)  
*Compute matrix forms of all OEP's within all OEP types.*
- virtual void **compute** (const std::string &oepType)=0  
*Compute matrix forms of all OEP's within a specified OEP type.*
- virtual void **compute\_3D** (const std::string &oepType, const double &x, const double &y, const double &z, std::shared\_ptr< psi::Vector > &v)=0  
*Compute value of potential in point x, y, z and save at v.*
- std::shared\_ptr< **OEPotential3D** < **OEPotential** > **make\_oeps3d** (const std::string &oepType)  
*Create 3D vector field with OEP.*
- virtual void **write\_cube** (const std::string &oepType, const std::string &fileName)  
*Write potential to a cube file.*
- 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.*
- **OEPType** **oep** (const std::string &oepType) const  
*Retrieve the potentials.*
- SharedMatrix **matrix** (const std::string &oepType) const  
*Retrieve the potentials in a matrix form.*
- SharedWavefunction **wfn** () const

*Retrieve wavefunction object.*

- void **set\_name** (const std::string &name)
- virtual void **print\_header** () const =0

## Static Public Member Functions

- static std::shared\_ptr  
< [OEPotential](#) > **build** (const std::string &category, SharedWavefunction [wfn](#), Options &options)  
*Build fully ESP-based OEP object.*
- static std::shared\_ptr  
< [OEPotential](#) > **build** (const std::string &category, SharedWavefunction [wfn](#), SharedBasisSet auxiliary, SharedBasisSet intermediate, Options &options)  
*Build general OEP object.*

## Protected Attributes

- Options [options\\_](#)  
*Psi4 options.*
- SharedWavefunction [wfn\\_](#)  
*Wavefunction.*
- SharedBasisSet [primary\\_](#)  
*Primary Basis set.*
- SharedBasisSet [auxiliary\\_](#)  
*Auxiliary Basis set.*
- SharedBasisSet [intermediate\\_](#)  
*Intermediate Basis set.*
- std::string [name\\_](#)  
*Name of this OEP;*
- std::map< std::string, [OEType](#) > [oepTypes\\_](#)  
*Types of OEP's within the scope of this object.*
- 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  
< [oepdev::PotentialInt](#) > [potInt\\_](#)  
*One-electron potential shared pointer.*
- std::shared\_ptr< psi::Matrix > [cOcc\\_](#)  
*Occupied orbitals.*
- std::shared\_ptr< psi::Matrix > [cVir\\_](#)  
*Virtual orbitals.*

### 17.40.1 Detailed Description

Generalized One-Electron Potential: Abstract base.

Manages OEP's in matrix and 3D forms. Available OEP categories:

- ELECTROSTATIC ENERGY
- REPULSION ENERGY
- CHARGE TRANSFER ENERGY
- EET COUPLING CONSTANT

### 17.40.2 Constructor & Destructor Documentation

#### 17.40.2.1 OEPotential::OEPotential ( SharedWavefunction *wfn*, Options & *options* )

Fully ESP-based OEP object.

Parameters

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

#### 17.40.2.2 OEPotential::OEPotential ( SharedWavefunction *wfn*, SharedBasisSet *auxiliary*, SharedBasisSet *intermediate*, Options & *options* )

General OEP object.

Parameters

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

### 17.40.3 Member Function Documentation

#### 17.40.3.1 std::shared\_ptr< OEPotential > OEPotential::build ( const std::string & *category*, SharedWavefunction *wfn*, Options & *options* ) [static]

Build fully ESP-based OEP object.

Parameters

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

#### 17.40.3.2 std::shared\_ptr< OEPotential > OEPotential::build ( const std::string & *category*, SharedWavefunction *wfn*, SharedBasisSet *auxiliary*, SharedBasisSet *intermediate*, Options & *options* ) [static]

Build general OEP object.

## Parameters

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

17.40.3.3 `std::shared_ptr< OEPotential3D< OEPotential > > OEPotential::make_oeps3d ( const std::string & oepType )`

Create 3D vector field with OEP.

## Parameters

<i>oepType</i>	- type of OEP. ESP-based OEP is assumed.
----------------	--

## Returns

Vector field 3D with the OEP values.

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

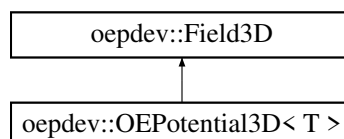
- [oepdev/liboep/oep.h](#)
- [oepdev/liboep/oep\\_base.cc](#)

## 17.41 oepdev::OEPotential3D< T > Class Template Reference

Class template for OEP 3D fields.

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

Inheritance diagram for `oepdev::OEPotential3D< T >`:



### Public Member Functions

- [OEPotential3D](#) (const int &ndim, const int &np, const double &padding, std::shared\_ptr< T > oep, const std::string &oepType)  
*Construct random spherical collection of 3D field of type T.*
- [OEPotential3D](#) (const int &ndim, 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 3D field of type T.*
- virtual [~OEPotential3D](#) ()  
*Destructor.*
- virtual std::shared\_ptr< psi::Vector > [compute\\_xyz](#) (const double &x, const double &y, const double &z)  
*Compute a value of 3D 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 3D field type stored in instance of T*

## Additional Inherited Members

### 17.41.1 Detailed Description

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

Class template for OEP 3D 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,
                   std::shared_ptr<psi::Vector> &v);

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

with the `descriptor` of a certain 3D field type, `x`, `y`, `z` the points in 3D space in which the scalar or vector 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

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

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

- `oepdev/lib3d/space3d.h`

## 17.42 oepdev::OEType Struct Reference

Container to handle the type of One-Electron Potentials.

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

## Public Attributes

- `std::string name`  
*Name of this type of OEP.*
- `bool is_density_fitted`  
*Is this OEP DF-based?*
- `int n`  
*Number of OEP's within a type.*

- SharedMatrix [matrix](#)  
*All OEP's of this type gathered in a matrix form.*
- SharedDMTPole [dmtp](#)  
*Distributed Multipole Object.*

### 17.42.1 Detailed Description

Container to handle the type of One-Electron Potentials.

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

- oepdev/liboep/[oep.h](#)

## 17.43 oepdev::PerturbCharges Struct Reference

Structure to hold perturbing charges.

```
#include <scf_perturb.h>
```

### Public Attributes

- `std::vector< double >` [charges](#)  
*Vector of charge values.*
- `std::vector< std::shared_ptr  
< psi::Vector > >` [positions](#)  
*Vector of charge position vectors.*

### 17.43.1 Detailed Description

Structure to hold perturbing charges.

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

- oepdev/libutil/[scf\\_perturb.h](#)

## 17.44 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/lib3d/space3d.h

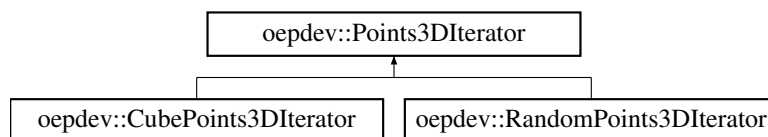


## 17.45 oepdev::Points3DIterator Class Reference

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_`

### 17.45.1 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.

### 17.45.2 Constructor & Destructor Documentation

#### 17.45.2.1 `oepdev::Points3DIterator::Points3DIterator ( const int & np )`

Plain constructor. Initializes the abstract features.

Parameters

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

### 17.45.3 Member Function Documentation

#### 17.45.3.1 `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

17.45.3.2 `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

17.45.3.3 `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:

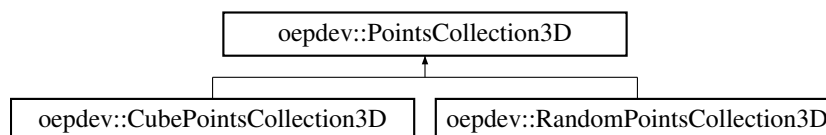
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

## 17.46 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*

### 17.46.1 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.

### 17.46.2 Constructor & Destructor Documentation

#### 17.46.2.1 oepdev::PointsCollection3D::PointsCollection3D ( Collection collectionType, int & np )

Initialize abstract features.

Parameters

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

### 17.46.3 Member Function Documentation

17.46.3.1 `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

17.46.3.2 `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

17.46.3.3 `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/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

## 17.47 oepdev::PolarGEFactory Class Reference

Polarization GEFP Factory. Abstract Base.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::PolarGEFactory:



## Public Member Functions

- **PolarGEFactory** (std::shared\_ptr< psi::Wavefunction > **wfn**, psi::Options &opt)  
*Construct from Psi4 options.*
- virtual **~PolarGEFactory** ()  
*Destruct.*
- virtual std::shared\_ptr< **GenEffPar** > **compute** (void)=0  
*Compute the density matrix susceptibility tensors.*

## Protected Member Functions

- std::shared\_ptr< psi::Vector > **draw\_field** ()  
*Randomly draw electric field value.*
- double **draw\_charge** ()  
*Randomly draw charge value.*
- std::shared\_ptr< **oepdev::RHFPerturbed** > **perturbed\_state** (const std::shared\_ptr< psi::Vector > &field)  
*Solve SCF equations to find perturbed state due to uniform electric field.*
- std::shared\_ptr< **oepdev::RHFPerturbed** > **perturbed\_state** (const std::shared\_ptr< psi::Vector > &pos, const double &charge)  
*Solve SCF equations to find perturbed state due to point charge.*
- std::shared\_ptr< **oepdev::RHFPerturbed** > **perturbed\_state** (const std::shared\_ptr< psi::Matrix > &charges)  
*Solve SCF equations to find perturbed state due to set of point charges.*
- std::shared\_ptr< psi::Vector > **field\_due\_to\_charges** (const std::shared\_ptr< psi::Matrix > &charges, const double &x, const double &y, const double &z)  
*Evaluate electric field at point (x,y,z) due to point charges.*
- std::shared\_ptr< psi::Vector > **field\_due\_to\_charges** (const std::shared\_ptr< psi::Matrix > &charges, const std::shared\_ptr< psi::Vector > &pos)
- std::shared\_ptr< psi::Matrix > **field\_gradient\_due\_to\_charges** (const std::shared\_ptr< psi::Matrix > &charges, const double &x, const double &y, const double &z)  
*Evaluate electric field gradient at point (x,y,z) due to point charges.*
- std::shared\_ptr< psi::Matrix > **field\_gradient\_due\_to\_charges** (const std::shared\_ptr< psi::Matrix > &charges, const std::shared\_ptr< psi::Vector > &pos)

## Additional Inherited Members

### 17.47.1 Detailed Description

Polarization GEFP Factory. Abstract Base.

Basic interface for the polarization density matrix susceptibility parameters.

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

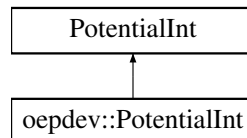
- [oepdev/libgefp/gefp.h](#)
- [oepdev/libgefp/gefp\\_polar\\_base.cc](#)

## 17.48 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::PotentilInt.*
- [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.*

### 17.48.1 Detailed Description

Computes potential integrals.

### 17.48.2 Constructor & Destructor Documentation

**17.48.2.1** 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::PotentilInt.

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

**17.48.2.2** 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

17.48.2.3 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 q
<i>y</i>	- y coordinate of q
<i>z</i>	- z coordinate of q
<i>q</i>	- value of the probe charge
<i>deriv</i>	- derivative level

## 17.48.3 Member Function Documentation

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

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

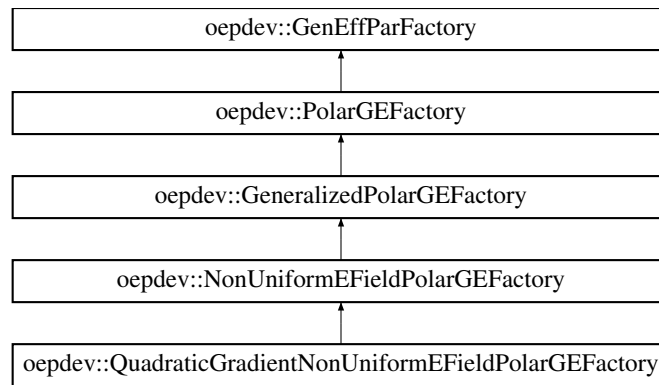
- oepdev/libpsi/[potential.h](#)
- oepdev/libpsi/potential.cc

## 17.49 oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory:



## Public Member Functions

- **QuadraticGradientNonUniformEFieldPolarGEFactory** (std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- void [compute\\_gradient](#) (int i, int j)  
*Compute Gradient vector associated with the i-th and j-th basis set function.*
- void [compute\\_hessian](#) (void)  
*Compute Hessian matrix (independent on the parameters)*

## Additional Inherited Members

### 17.49.1 Detailed Description

Polarization GEFP Factory with Least-Squares Parameterization.

Implements the generalized density matrix susceptibility model of the form

$$\delta D_{\alpha\beta} \approx \sum_i \left\{ \mathbf{B}_{i;\alpha\beta}^{(10)} \cdot \mathbf{F}(\mathbf{r}_i) + \mathbf{B}_{i;\alpha\beta}^{(20)} : \mathbf{F} \otimes \mathbf{F}(\mathbf{r}_i) + \mathbf{B}_{i;\alpha\beta}^{(01)} : \nabla_i \otimes \mathbf{F}(\mathbf{r}_i) \right\}$$

where:

- $\mathbf{B}_{i;\alpha\beta}^{(10)}$  is the density matrix dipole polarizability
- $\mathbf{B}_{i;\alpha\beta}^{(20)}$  is the density matrix dipole-dipole hyperpolarizability
- $\mathbf{B}_{i;\alpha\beta}^{(01)}$  is the density matrix quadrupole polarizability all defined for the distributed site at  $\mathbf{r}_i$ .

## Note

This model is not available now and probably will be deprecated in the future.

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

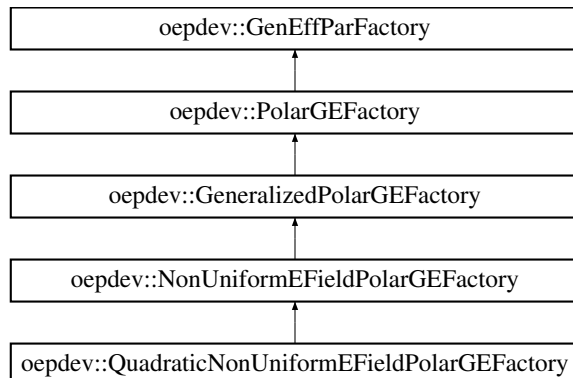
- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp\_polar\_nonuniform\_field\_2\_grad\_1.cc

## 17.50 oepdev::QuadraticNonUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::QuadraticNonUniformEFieldPolarGEFactory:



### Public Member Functions

- **QuadraticNonUniformEFieldPolarGEFactory** (std::shared\_ptr< psi::Wavefunction > wfn, psi::Options &opt)
- void **compute\_gradient** (int i, int j)  
*Compute Gradient vector associated with the i-th and j-th basis set function.*
- void **compute\_hessian** (void)  
*Compute Hessian matrix (independent on the parameters)*

### Additional Inherited Members

#### 17.50.1 Detailed Description

Polarization GEFP Factory with Least-Squares Parameterization.

Implements the generalized density matrix susceptibility model of the form

$$\delta D_{\alpha\beta} \approx \sum_i \left\{ \mathbf{B}_{i;\alpha\beta}^{(10)} \cdot \mathbf{F}(\mathbf{r}_i) + \mathbf{B}_{i;\alpha\beta}^{(20)} : \mathbf{F}(\mathbf{r}_i) \otimes \mathbf{F}(\mathbf{r}_i) \right\}$$

where:

- $\mathbf{B}_{i;\alpha\beta}^{(10)}$  is the density matrix dipole polarizability
- $\mathbf{B}_{i;\alpha\beta}^{(20)}$  is the density matrix dipole-dipole hyperpolarizability all defined for the distributed site at  $\mathbf{r}_i$ .

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

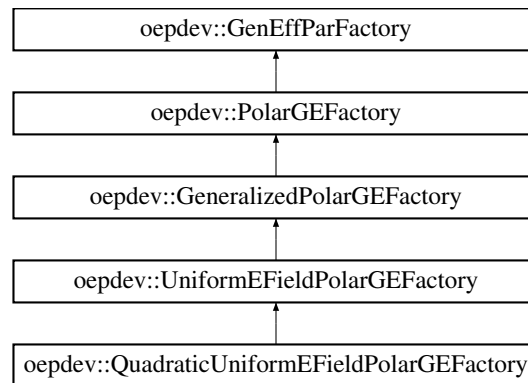
- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp\_polar\_nonuniform\_field\_2.cc

## 17.51 oepdev::QuadraticUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::QuadraticUniformEFieldPolarGEFactory:



## Public Member Functions

- **QuadraticUniformEFieldPolarGEFactory** (std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- void [compute\\_gradient](#) (int i, int j)  
*Compute Gradient vector associated with the i-th and j-th basis set function.*
- void [compute\\_hessian](#) (void)  
*Compute Hessian matrix (independent on the parameters)*

## Additional Inherited Members

### 17.51.1 Detailed Description

Polarization GEFP Factory with Least-Squares Parameterization.

Implements the generalized density matrix susceptibility model of the form

$$\delta D_{\alpha\beta} \approx \mathbf{B}_{\alpha\beta}^{(10)} \cdot \mathbf{F} + \mathbf{B}_{\alpha\beta}^{(20)} : \mathbf{F} \otimes \mathbf{F}$$

where:

- $\mathbf{B}_{\alpha\beta}^{(10)}$  is the density matrix dipole polarizability
- $\mathbf{B}_{\alpha\beta}^{(20)}$  is the density matrix dipole-dipole hyperpolarizability

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

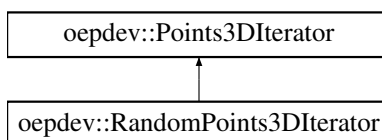
- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp\_polar\_uniform\_field\_2.cc

## 17.52 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

### 17.52.1 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:

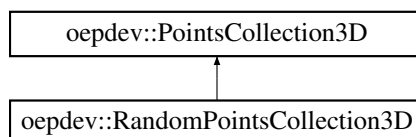
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

## 17.53 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

#### 17.53.1 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:

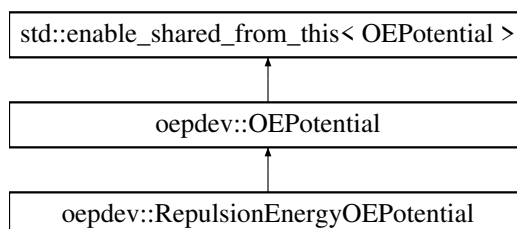
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

## 17.54 oepdev::RepulsionEnergyOEPotential Class Reference

Generalized One-Electron Potential for Pauli Repulsion Energy.

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

Inheritance diagram for oepdev::RepulsionEnergyOEPotential:



## Public Member Functions

- **RepulsionEnergyOEPotential** (SharedWavefunction [wfn](#), SharedBasisSet auxiliary, SharedBasisSet intermediate, Options &options)
- **RepulsionEnergyOEPotential** (SharedWavefunction [wfn](#), Options &options)
- virtual void [compute](#) (const std::string &oepType) override  
*Compute matrix forms of all OEP's within a specified OEP type.*
- virtual void [compute\\_3D](#) (const std::string &oepType, const double &x, const double &y, const double &z, std::shared\_ptr< psi::Vector > &v) override  
*Compute value of potential in point x, y, z and save at v.*
- virtual void **print\_header** () const override

## Additional Inherited Members

### 17.54.1 Detailed Description

Generalized One-Electron Potential for Pauli Repulsion Energy.

Contains the following OEP types:

- `Murrell-etal.S1`
- `Otto-Ladik.S2`

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

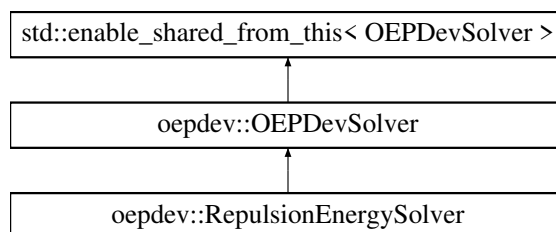
- `oepdev/liboep/oep.h`
- `oepdev/liboep/oep_energy_pauli.cc`

## 17.55 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

### 17.55.1 Detailed Description

Compute the Pauli-Repulsion interaction energy between unperturbed wavefunctions.

The implemented methods are shown below *Note*:

Keyword	Method Description
<b>Benchmark Methods</b>	
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 (1967).
OTTO_LADIK	Approximate Pauli Repulsion energy at HF level from Otto and Ladik (1975).
EFP2	Approximate Pauli Repulsion energy at HF level from EFP2 model.
<b>OEP-Based Methods</b>	
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 17.3: Methods available in the Solver

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

Below the detailed description of the implemented equations is given for each of the above provided methods. In the formulae across, it is assumed that the orbitals are real. The Coulomb notation for electron repulsion integrals (ERI's) is adopted; i.e,

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

Greek subscripts denote basis set orbitals whereas Italic subscripts denote the occupied molecular orbitals.

### 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 few approximate methods described below (Otto-Ladik and EFP2 methods).

## 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

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

Thus derived repulsion energy is invariant with respect to transformation of molecular orbitals, similarly as Hayes--Stone's method and density-based method. By using OEP technique, the above theory can be exactly re-cast *without* any further approximations.

#### Approximate Pauli Repulsion energy at HF level from Otto and Ladik (1975).

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

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

whereas the second-order term is

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

Thus derived repulsion energy is *not* invariant with respect to transformation of molecular orbitals, in contrast to Hayes-Stone's method and density-based method. It was shown that good results are obtained when using localized molecular orbitals, whereas using canonical molecular orbitals brings poor results. By using OEP technique, the above theory can be exactly re-cast *without* any further approximations.

#### Approximate Pauli Repulsion energy at HF level from Jensen and Gordon (1996).

The Pauli repulsion energy used within the EFP2 approach 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

$$E^{\text{Rep}}(\mathcal{O}(S)) = -2 \sum_{a \in A} \sum_{b \in B} S_{ab} \left\{ \sum_{c \in A} F_{ac}^A S_{cb} + \sum_{d \in B} F_{bd}^B S_{da} - 2T_{ab} \right\}$$

whereas the second-order term is

$$E^{\text{Rep}}(\mathcal{O}(S^2)) = 2 \sum_{a \in A} \sum_{b \in B} S_{ab}^2 \left\{ \sum_{x \in A} \frac{-Z_x}{R_{xb}} + \sum_{y \in B} \frac{-Z_y}{R_{ya}} + \sum_{c \in A} \frac{2}{R_{bc}} + \sum_{d \in B} \frac{2}{R_{ad}} - \frac{1}{R_{ab}} \right\}$$

Thus derived repulsion energy is *not* invariant with respect to transformation of molecular orbitals, in contrast to Hayes-Stone's method and density-based method. It was shown that good results are obtained when using localized molecular orbitals, whereas using canonical molecular orbitals brings poor results.

In EFP2, exchange energy is approximated by spherical Gaussian approximation (SGO). The result of this is the following formula for the exchange energy:

$$E^{\text{Ex}} \approx -4 \sum_{a \in A} \sum_{b \in B} \sqrt{\frac{-2 \ln |S_{ab}|}{\pi}} \frac{S_{ab}^2}{R_{ab}}$$

In all the above formulas,  $R_{ij}$  are distances between position vectors of  $i$ th and  $j$ th point. The LMO centroids are defined by

$$\mathbf{r}_a = (a|\mathbf{r}|a)$$

where  $a$  denotes the occupied molecular orbital.

## OEP-Based Methods

The Murrell et al's theory of Pauli repulsion for S-1 term and the Otto-Ladik's theory for S-2 term is here re-cast by introducing OEP's. The S-1 term is expressed via DF-OEP, whereas the S-2 term via ESP-OEP.

### S-1 term (Murrell et al.)

The OEP reduction without any approximations leads to the following formula

$$E^{\text{Rep}}(\mathcal{O}(S^1)) = -2 \sum_{a \in A} \sum_{b \in B} S_{ab} \left\{ \sum_{\xi \in A} S_{b\xi} G_{\xi a}^A + \sum_{\eta \in B} S_{a\eta} G_{\eta b}^B \right\}$$

where the OEP matrices are given as

$$G_{\xi a}^A = \sum_{\xi' \in A} [S^{-1}]_{\xi\xi'} \sum_{\alpha \in A} \left\{ C_{\alpha a} V_{\alpha\xi'}^A + \sum_{\mu \nu \in A} [2C_{\alpha a} D_{\mu\nu} - C_{\nu a} D_{\alpha\mu}] (\alpha\xi' | \mu\nu) \right\}$$

and analogously for molecule *B*. Here, the nuclear attraction integrals are denoted by  $V_{\alpha\xi'}^A$ .

### S-2 term (Otto-Ladik)

After the OEP reduction, this contribution under Otto-Ladik approximation has the following form:

$$E^{\text{Rep}}(\mathcal{O}(S^2)) = 2 \sum_{a \in A} \sum_{b \in B} S_{ab}^2 \left\{ \sum_{x \in A} q_{xa} V_{bb}^{(x)} + \sum_{y \in B} q_{yb} V_{aa}^{(y)} \right\}$$

where the ESP charges associated with each occupied molecular orbital reproduce the *effective potential* of molecule in question, i.e.,

$$\sum_{x \in A} \frac{q_{xa}}{|\mathbf{r} - \mathbf{r}_x|} \cong v_a^A(\mathbf{r})$$

where the potential is given by

$$v_a^A(\mathbf{r}) = \sum_{x \in A} \frac{-Z_x}{|\mathbf{r} - \mathbf{r}_x|} + 2 \sum_{c \in A} \int \frac{\phi_c(\mathbf{r}') \phi_c(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' - \frac{1}{2} \int \frac{\phi_a(\mathbf{r}') \phi_a(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}'$$

## 17.55.2 Member Function Documentation

**17.55.2.1** `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](#).

**17.55.2.2** `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:

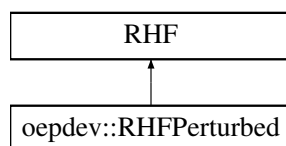
- [oepdev/libsolver/solver.h](#)
- [oepdev/libsolver/solver\\_energy\\_pauli.cc](#)

## 17.56 oepdev::RHPerturbed Class Reference

RHF theory under electrostatic perturbation.

```
#include <scf_perturb.h>
```

Inheritance diagram for `oepdev::RHPerturbed`:



### Public Member Functions

- [RHPerturbed](#) (std::shared\_ptr< psi::Wavefunction > ref\_wfn, std::shared\_ptr< psi::SuperFunctional > functional)  
*Build from wavefunction and superfunctional.*
- [RHPerturbed](#) (std::shared\_ptr< psi::Wavefunction > ref\_wfn, std::shared\_ptr< psi::SuperFunctional > functional, psi::Options &options, std::shared\_ptr< psi::PSIO > psio)  
*Build from wavefunction and superfunctional + options and psio.*
- virtual [~RHPerturbed](#) ()  
*Clear memory.*
- virtual double [compute\\_energy](#) ()  
*Compute total energy.*
- virtual void [set\\_perturbation](#) (std::shared\_ptr< psi::Vector > field)  
*Perturb the system with external electric field.*
- virtual void [set\\_perturbation](#) (const double &fx, const double &fy, const double &fz)  
*This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.*
- virtual void [set\\_perturbation](#) (std::shared\_ptr< psi::Vector > position, const double &charge)  
*Perturb the system with a point charge.*
- virtual void [set\\_perturbation](#) (const double &rx, const double &ry, const double &rz, const double &charge)  
*This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.*
- std::shared\_ptr< psi::Matrix > [Vpert](#) () const  
*Get a copy of the perturbation potential one-electron matrix.*
- double [nuclear\\_interaction\\_energy](#) () const  
*Get the interaction energy of the nuclei with the perturbing potential.*

## Protected Member Functions

- virtual void [perturb\\_Hcore](#) ()  
*Add the electrostatic perturbation to the Hcore matrix.*

## Protected Attributes

- std::shared\_ptr< psi::Vector > [perturbField\\_](#)  
*Perturbing electric field.*
- std::shared\_ptr< [PerturbCharges](#) > [perturbCharges\\_](#)  
*Perturbing charges.*
- std::shared\_ptr< psi::Matrix > [Vpert\\_](#)  
*Perturbation potential one-electron matrix.*
- double [nuclearInteractionEnergy\\_](#)  
*Electrostatic interaction energy due to nuclei.*

### 17.56.1 Detailed Description

RHF theory under electrostatic perturbation.

Compute RHF wavefunction under the following conditions:

- external uniform electric field
- set of point charges The mixed conditions can also be used.

#### Theory

The electrostatic perturbation is here understood as a distribution of external (generally non-uniform) electric field. It is assumed that this perturbation is one-electron in nature. Therefore, the one-electron Hamiltonian is changed according to the following

$$\mathbf{H}^{\text{core}} \rightarrow \mathbf{H}^{\text{core}} + \sum_n q_n \mathbf{V}^{(n)} - \mathbb{M} \cdot \mathbf{F}$$

where  $q_n$  is the external classical point charge,  $\mathbf{V}^{(n)}$  is the associated matrix of potential integrals,  $\mathbb{M}$  is the vector of dipole integrals and  $\mathbf{F}$  is an external uniform electric field. The total energy is then computed by performing an SCF procedure on the above one-electron Hamiltonian. The contribution due to nuclei is included, i.e.,

$$E_{\text{Nuc}} \rightarrow E_{\text{Nuc-Nuc}} + \sum_{In} \frac{q_n Z_I}{r_{In}} - \mu_{\text{Nuc}} \cdot \mathbf{F}$$

where  $\mu_{\text{Nuc}}$  is the nuclear dipole moment and  $Z_I$  is the atomic number of the  $I$ th nucleus. It is added in the nuclear repulsion energy  $E_{\text{Nuc-Nuc}}$  (note that the resulting energy can be negative as well depending on the electric field direction and configuration of point charges).

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

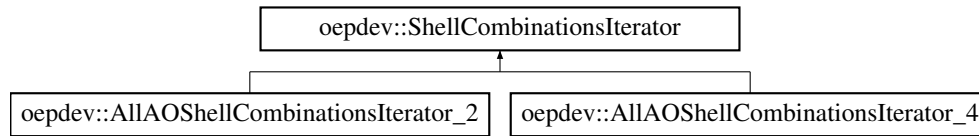
- oepdev/libutil/[scf\\_perturb.h](#)
- oepdev/libutil/[scf\\_perturb.cc](#)

## 17.57 oepdev::ShellCombinationsIterator Class Reference

Iterator for Shell Combinations. Abstract Base.

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

Inheritance diagram for oepdev::ShellCombinationsIterator:



## Public Member Functions

- [ShellCombinationsIterator](#) (int [nshell](#))  
*Constructor.*
- virtual [~ShellCombinationsIterator](#) ()  
*Destructor.*
- virtual void [first](#) (void)=0  
*First iteration.*
- virtual void [next](#) (void)=0  
*Next iteration.*
- virtual std::shared\_ptr  
< psi::BasisSet > [bs\\_1](#) (void) const  
*Grab the basis set of axis 1.*
- virtual std::shared\_ptr  
< psi::BasisSet > [bs\\_2](#) (void) const  
*Grab the basis set of axis 2.*
- virtual std::shared\_ptr  
< psi::BasisSet > [bs\\_3](#) (void) const  
*Grab the basis set of axis 3.*
- virtual std::shared\_ptr  
< psi::BasisSet > [bs\\_4](#) (void) const  
*Grab the basis set of axis 4.*
- virtual int [P](#) (void) const  
*Grab the current shell P index.*
- virtual int [Q](#) (void) const  
*Grab the current shell Q index.*
- virtual int [R](#) (void) const  
*Grab the current shell R index.*
- virtual int [S](#) (void) const  
*Grab the current shell S index.*
- virtual bool [is\\_done](#) (void)  
*Return status of an iterator.*
- virtual const int [nshell](#) (void) const  
*Return number of shells this iterator is for.*
- virtual std::shared\_ptr  
< [AOIntegralsIterator](#) > [ao\\_iterator](#) (std::string mode="ALL") const
- virtual void [compute\\_shell](#) (std::shared\_ptr< [oepdev::TwoBodyAOInt](#) > tei) const =0
- virtual void [compute\\_shell](#) (std::shared\_ptr< psi::TwoBodyAOInt > tei) const =0

## Static Public Member Functions

- static std::shared\_ptr  
< [ShellCombinationsIterator](#) > [build](#) (const [IntegralFactory](#) &ints, std::string mode="ALL", int [nshell](#)=4)  
*Build shell iterator from [oepdev::IntegralFactory](#).*
- static std::shared\_ptr  
< [ShellCombinationsIterator](#) > [build](#) (std::shared\_ptr< [IntegralFactory](#) > ints, std::string mode="ALL", int [nshell](#)=4)  
*This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.*
- static std::shared\_ptr  
< [ShellCombinationsIterator](#) > [build](#) (const psi::IntegralFactory &ints, std::string mode="ALL", int [nshell](#)=4)  
*Build shell iterator from [psi::IntegralFactory](#).*
- static std::shared\_ptr  
< [ShellCombinationsIterator](#) > [build](#) (std::shared\_ptr< psi::IntegralFactory > ints, std::string mode="ALL", int [nshell](#)=4)  
*This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.*

## Protected Attributes

- SharedBasisSet [bs\\_1\\_](#)  
*Basis set of axis 1.*
- SharedBasisSet [bs\\_2\\_](#)  
*Basis set of axis 2.*
- SharedBasisSet [bs\\_3\\_](#)  
*Basis set of axis 3.*
- SharedBasisSet [bs\\_4\\_](#)  
*Basis set of axis 4.*
- const int [nshell\\_](#)  
*Number of shells this iterator is for.*
- bool [done](#)  
*Status of an iterator.*

### 17.57.1 Detailed Description

Iterator for Shell Combinations. Abstract Base.

Date

2018/03/01 17:22:00

### 17.57.2 Constructor & Destructor Documentation

#### 17.57.2.1 ShellCombinationsIterator::ShellCombinationsIterator ( int *nshell* )

Constructor.

Parameters

---

<i>nshell</i>	- number of shells this iterator is for
---------------	---

### 17.57.3 Member Function Documentation

17.57.3.1 `std::shared_ptr< AOIntegralsIterator > ShellCombinationsIterator::ao_iterator ( std::string mode = "ALL" ) const` [virtual]

Make an AO integral iterator based on current shell

Parameters

<i>mode</i>	- either "ALL" or "UNIQUE" (iterate over all or unique integrals)
-------------	---

Returns

iterator over AO integrals

17.57.3.2 `std::shared_ptr< ShellCombinationsIterator > ShellCombinationsIterator::build ( const IntegralFactory & ints, std::string mode = "ALL", int nshell = 4 )` [static]

Build shell iterator from [oepdev::IntegralFactory](#).

Parameters

<i>ints</i>	- integral factory
<i>mode</i>	- mode of iteration (either ALL or UNIQUE)
<i>nshell</i>	- number of shells to iterate through

Returns

shell iterator

Examples:

[example\\_integrals\\_iter.cc](#).

17.57.3.3 `std::shared_ptr< ShellCombinationsIterator > ShellCombinationsIterator::build ( const psi::IntegralFactory & ints, std::string mode = "ALL", int nshell = 4 )` [static]

Build shell iterator from `psi::IntegralFactory`.

Parameters

<i>ints</i>	- integral factory
<i>mode</i>	- mode of iteration (either ALL or UNIQUE)
<i>nshell</i>	- number of shells to iterate through

Returns

shell iterator

17.57.3.4 `void ShellCombinationsIterator::compute_shell ( std::shared_ptr< oepdev::TwoBodyAOInt > tei ) const` [pure virtual]

Compute integrals in a current shell. Works both for [oepdev::TwoBodyAOInt](#) and `psi::TwoBodyAOInt`



## Parameters

<i>tei</i>	- two body integral object
------------	----------------------------

Implemented in [oepdev::AllAOShellCombinationsIterator\\_2](#), and [oepdev::AllAOShellCombinationsIterator\\_4](#).

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

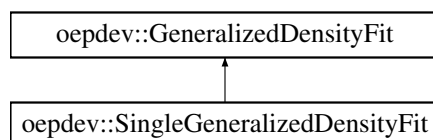
- [oepdev/libutil/integrals\\_iter.h](#)
- [oepdev/libutil/integrals\\_iter.cc](#)

## 17.58 oepdev::SingleGeneralizedDensityFit Class Reference

Generalized Density Fitting Scheme - Single Fit.

```
#include <oep_gdf.h>
```

Inheritance diagram for oepdev::SingleGeneralizedDensityFit:



### Public Member Functions

- **SingleGeneralizedDensityFit** (std::shared\_ptr< psi::BasisSet > bs\_auxiliary, std::shared\_ptr< psi::Matrix > v\_vector)
- std::shared\_ptr< psi::Matrix > [compute](#) (void)  
*Perform the generalized density fit.*

### Additional Inherited Members

#### 17.58.1 Detailed Description

Generalized Density Fitting Scheme - Single Fit.

The density fitting map projects the OEP onto the auxiliary, nearly complete basis set space through application of the resolution of identity. Refer to [density fitting in complete space](#) for more details.

#### 17.58.2 Determination of the OEP matrix

Coefficients **G** are computed by using the following relation

$$\mathbf{G}^{(i)} = \mathbf{v}^{(i)} \cdot \mathbf{S}^{-1}$$

where

$$S_{\xi\eta} = (\xi|\eta)$$

$$v_{\xi}^{(i)} = (\xi|\hat{v}i)$$

In the above,  $\int$  denotes the single integration over electron coordinate, i.e.,

$$(a|b) \equiv \int d\mathbf{r} \phi_a^*(\mathbf{r}) \phi_b(\mathbf{r})$$

whereas the spatial form of the potential operator  $\hat{v}$  can be expressed by

$$v(\mathbf{r}) \equiv \int d\mathbf{r}' \frac{\rho(\mathbf{r}')}{|\mathbf{r}' - \mathbf{r}|}$$

with  $\rho(\mathbf{r})$  being the effective one-electron density associated with  $\hat{v}$ .

### 17.58.3 Member Function Documentation

#### 17.58.3.1 `std::shared_ptr< psi::Matrix > SingleGeneralizedDensityFit::compute ( void ) [virtual]`

Perform the generalized density fit.

##### Returns

The OEP coefficients  $G_{\xi i}$

Implements [oepdev::GeneralizedDensityFit](#).

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

- [oepdev/liboep/oep\\_gdf.h](#)
- [oepdev/liboep/oep\\_gdf.cc](#)

## 17.59 oepdev::GeneralizedPolarGEFactory::StatisticalSet Struct Reference

A structure to handle statistical data.

```
#include <gefp.h>
```

### Public Attributes

- `std::vector< double >` [InducedInteractionEnergySet](#)  
*Interaction energy set.*
- `std::vector< std::shared_ptr< psi::Matrix > >` [DensityMatrixSet](#)  
*Density matrix set.*
- `std::vector< std::shared_ptr< psi::Vector > >` [InducedDipoleSet](#)  
*Induced dipole moment set.*
- `std::vector< std::shared_ptr< psi::Vector > >` [InducedQuadrupoleSet](#)  
*Induced quadrupole moment set.*
- `std::vector< std::shared_ptr< psi::Matrix > >` [JKMatrixSet](#)  
*Sum of J and K matrix set.*

#### 17.59.1 Detailed Description

A structure to handle statistical data.

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

- [oepdev/libgefp/gefp.h](#)

## 17.60 oepdev::test::Test Class Reference

Manages test routines.

```
#include <test.h>
```

### Public Member Functions

- [Test](#) (std::shared\_ptr< psi::Wavefunction > wfn, psi::Options &options)  
*Construct the tester.*
- [~Test](#) ()  
*Destructor.*
- double [run](#) (void)  
*Peform the test.*

### Protected Member Functions

- double [test\\_basic](#) (void)  
*Test the basic functionalities of OEPDev.*
- double [test\\_cphf](#) (void)  
*Test the CPHF method.*
- double [test\\_dmatPol](#) (void)  
*Test the density matrix susceptibility ( $X = 1$ )*
- double [test\\_dmatPolX](#) (void)  
*Test the density matrix susceptibility.*
- double [test\\_eri\\_1\\_1](#) (void)  
*Test the oepdev::ERI\_1\_1 class against psi::ERI.*
- double [test\\_eri\\_2\\_2](#) (void)  
*Test the oepdev::ERI\_2\_2 class against psi::ERI.*
- double [test\\_eri\\_3\\_1](#) (void)  
*Test the oepdev::ERI\_3\_1 class against psi::ERI.*
- double [test\\_unitaryOptimizer](#) (void)  
*Test the oepdev::UnitaryOptimizer class.*
- double [test\\_unitaryOptimizer\\_4\\_2](#) (void)  
*Test the oepdev::UnitaryOptimizer\_4\_2 class.*
- double [test\\_scf\\_perturb](#) (void)  
*Test the oepdev::RHFPerturbed class.*
- double [test\\_camm](#) (void)  
*Test the oepdev::CAMM class.*
- double [test\\_dmtpl\\_energy](#) (void)  
*Test the oepdev::DMTP class for energy calculations.*
- double [test\\_custom](#) (void)  
*Test the custom code.*

### Protected Attributes

- std::shared\_ptr  
    < psi::Wavefunction > [wfn\\_](#)  
    *Wavefunction object.*
- psi::Options & [options\\_](#)  
    *Psi4 Options.*

### 17.60.1 Detailed Description

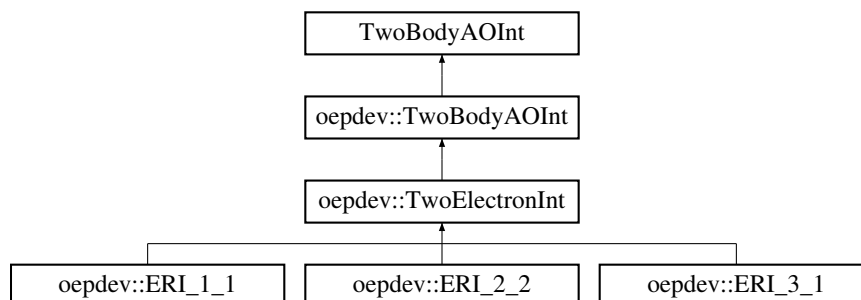
Manages test routines.

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

- oepdev/libtest/test.h
- oepdev/libtest/test.cc
- oepdev/libtest/test\_custom.cc

## 17.61 oepdev::TwoBodyAOInt Class Reference

Inheritance diagram for oepdev::TwoBodyAOInt:



### Public Member Functions

- virtual void [compute](#) (std::shared\_ptr< psi::Matrix > &result, int ibs1=0, int ibs2=2)  
*Compute two-body two-centre integral and put it into matrix.*
- virtual void [compute](#) (psi::Matrix &result, int ibs1=0, int ibs2=2)
- virtual size\_t [compute\\_shell](#) (int, int, int, int)=0
- virtual size\_t [compute\\_shell](#) (int, int, int)=0
- virtual size\_t [compute\\_shell](#) (int, int)=0
- virtual size\_t [compute\\_shell\\_deriv1](#) (int, int, int, int)=0
- virtual size\_t [compute\\_shell\\_deriv2](#) (int, int, int, int)=0
- virtual size\_t [compute\\_shell\\_deriv1](#) (int, int, int)=0
- virtual size\_t [compute\\_shell\\_deriv2](#) (int, int, int)=0
- virtual size\_t [compute\\_shell\\_deriv1](#) (int, int)=0
- virtual size\_t [compute\\_shell\\_deriv2](#) (int, int)=0

### Protected Member Functions

- **TwoBodyAOInt** (const [IntegralFactory](#) \*intsfactory, int deriv=0)
- **TwoBodyAOInt** (const [TwoBodyAOInt](#) &rhs)

### 17.61.1 Member Function Documentation

17.61.1.1 void oepdev::TwoBodyAOInt::compute ( std::shared\_ptr< psi::Matrix > & result, int ibs1 = 0, int ibs2 = 2 )  
[virtual]

Compute two-body two-centre integral and put it into matrix.

## Parameters

<i>result</i>	- matrix where to store (i  j) two-body integrals
<i>ibs1</i>	- first basis set axis
<i>ibs2</i>	- second basis set axis

17.61.1.2 void oepdev::TwoBodyAOInt::compute ( psi::Matrix & *result*, int *ibs1* = 0, int *ibs2* = 2 ) [virtual]

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

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

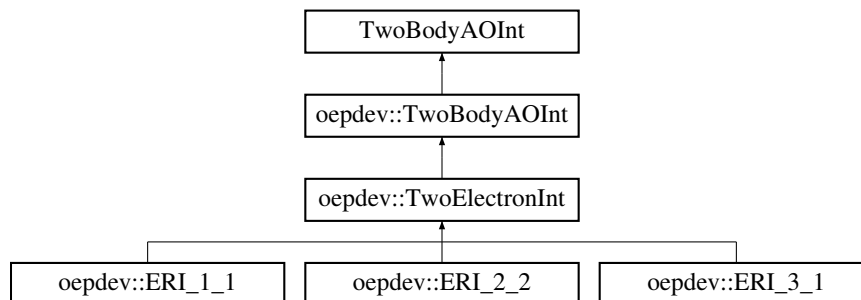
- oepdev/libpsi/integral.h
- oepdev/libpsi/integral.cc

## 17.62 oepdev::TwoElectronInt Class Reference

General Two Electron Integral.

```
#include <eri.h>
```

Inheritance diagram for oepdev::TwoElectronInt:



### Public Member Functions

- **TwoElectronInt** (const [IntegralFactory](#) \*integral, int deriv, bool use\_shell\_pairs)
- virtual size\_t [compute\\_shell](#) (int, int)  
Compute ERI's between 2 shells. Result is stored in buffer.
- virtual size\_t [compute\\_shell](#) (int, int, int)  
Compute ERI's between 3 shells. Result is stored in buffer.
- virtual size\_t [compute\\_shell](#) (int, int, int, int)  
Compute ERI's between 4 shells. Result is stored in buffer.
- virtual size\_t [compute\\_shell](#) (const psi::AOShellCombinationsIterator &)
- virtual size\_t [compute\\_shell\\_deriv1](#) (int, int)  
Compute first derivatives of ERI's between 2 shells.
- virtual size\_t [compute\\_shell\\_deriv2](#) (int, int)  
Compute second derivatives of ERI's between 2 shells.
- virtual size\_t [compute\\_shell\\_deriv1](#) (int, int, int)  
Compute first derivatives of ERI's between 3 shells.
- virtual size\_t [compute\\_shell\\_deriv2](#) (int, int, int)  
Compute second derivatives of ERI's between 3 shells.

- virtual size\_t [compute\\_shell\\_deriv1](#) (int, int, int, int)  
*Compute first derivatives of ERI's between 4 shells.*
- virtual size\_t [compute\\_shell\\_deriv2](#) (int, int, int, int)  
*Compute second derivatives of ERI's between 4 shells.*

## Protected Member Functions

- int [get\\_cart\\_am](#) (int am, int n, int x)  
*Get the angular momentum per Cartesian component.*
- double [get\\_R](#) (int N, int L, int M)  
*Get the (N,L,M)th McMurchie-Davidson coefficient.*
- virtual size\_t [compute\\_doublet](#) (int, int)  
*Computes the ERI's between three shells.*
- virtual size\_t [compute\\_triplet](#) (int, int, int)  
*Computes the ERI's between three shells.*
- virtual size\_t [compute\\_quartet](#) (int, int, int, int)  
*Computes the ERI's between four shells.*

## Protected Attributes

- const int [max\\_am\\_](#)  
*Maximum angular momentum.*
- const int [n\\_max\\_am\\_](#)  
*Maximum number of angular momentum functions.*
- psi::Fjt \* [fjt\\_](#)  
*Computes the fundamental: Boys function value at T for degree v.*
- bool [use\\_shell\\_pairs\\_](#)  
*Should we use shell pair information?*
- const double [cartMap\\_](#) [60]  
*Map of Cartesian components per each am.*
- const double [df\\_](#) [8]  
*Double factorial array.*
- double \* [mdh\\_buffer\\_R\\_](#)  
*Buffer for the McMurchie-Davidson-Hermite R coefficients.*

### 17.62.1 Detailed Description

General Two Electron Integral.

Implements the McMurchie-Davidson recursive scheme for all integral types. The integral can be defined for any number of Gaussian centres, thus it is not limited to 2-by-2 four-centre ERI. Currently implemented subtypes are:

- [oepdev::ERI\\_1\\_1](#) - 2-centre electron-repulsion integral (i|j)
- [oepdev::ERI\\_2\\_2](#) - 4-centre electron-repulsion integral (ij|kl)
- [oepdev::ERI\\_3\\_1](#) - 4-centre electron-repulsion integral (ijk|l)

See Also

[The Integral Package Library](#)

## 17.62.2 Member Function Documentation

17.62.2.1 `size_t oepdev::TwoElectronInt::compute_shell ( const psi::AOShellCombinationsIterator & shellIter )`  
`[virtual]`

Compute ERIs between 4 shells. Result is stored in buffer. Only for use with [ERI\\_2\\_2](#) and the same basis sets, otherwise shell pairs won't be compatible.

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

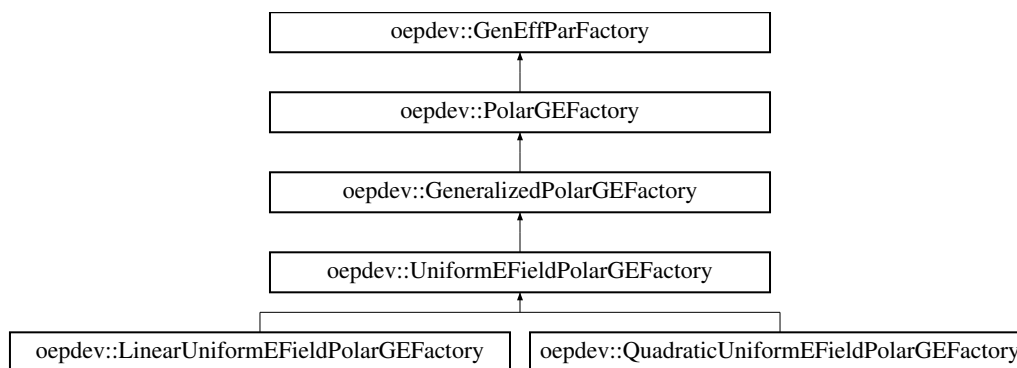
- oepdev/libints/[eri.h](#)
- oepdev/libints/[eri.cc](#)

## 17.63 oepdev::UniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::UniformEFieldPolarGEFactory:



### Public Member Functions

- **UniformEFieldPolarGEFactory** (std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- void [compute\\_samples](#) (void)  
*Compute samples of density matrices and select electric field distributions.*
- virtual void [compute\\_gradient](#) (int i, int j)=0  
*Compute Gradient vector associated with the i-th and j-th basis set function.*
- virtual void [compute\\_hessian](#) (void)=0  
*Compute Hessian matrix (independent on the parameters)*

### Additional Inherited Members

#### 17.63.1 Detailed Description

Polarization GEFP Factory with Least-Squares Parameterization.

Implements a class of density matrix susceptibility models for parameterization in the uniform electric field.

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

- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/[gefp\\_polar\\_uniform\\_base.cc](#)

## 17.64 oepdev::UnitaryOptimizer Class Reference

Find the optimum unitary matrix of quadratic matrix equation.

```
#include <unitary_optimizer.h>
```

### Public Member Functions

- [UnitaryOptimizer](#) (double \*R, double \*P, int n, double conv=1.0e-6, int maxiter=100, bool verbose=true)  
*Create from R and P matrices and optimization options.*
- [UnitaryOptimizer](#) (std::shared\_ptr< psi::Matrix > R, std::shared\_ptr< psi::Vector > P, double conv=1.0e-6, int maxiter=100, bool verbose=true)  
*Create from R and P matrices and optimization options.*
- [~UnitaryOptimizer](#) ()  
*Clear memory.*
- bool [maximize](#) ()  
*Run the minimization.*
- bool [minimize](#) ()  
*Run the maximization.*
- std::shared\_ptr< psi::Matrix > [X](#) ()  
*Get the unitary matrix (solution)*
- double \* [get\\_X](#) () const  
*Get the unitary matrix (pointer to solution)*
- double [Z](#) ()  
*Get the actual value of Z function.*
- bool [success](#) () const  
*Get the status of the optimization.*

### Protected Member Functions

- [UnitaryOptimizer](#) (int n, double conv, int maxiter, bool verbose)  
*Initialize the basic memory.*
- void [common\\_init](#) ()  
*Prepare the optimizer.*
- void [run](#) (const std::string &opt)  
*Run the optimization (intermediate interface)*
- void [optimize](#) (const std::string &opt)  
*Run the optimization (inner interface)*
- void [refresh](#) ()  
*Restore the initial state of the optimizer.*
- void [update\\_conv](#) ()  
*Update the convergence.*
- void [update\\_iter](#) ()  
*Update the iterates.*
- void [update\\_Z](#) ()  
*Update Z value.*
- void [update\\_RP](#) ()  
*Update R and P matrices.*
- void [update\\_X](#) ()  
*Update the solution matrix X.*
- double [eval\\_Z](#) (double \*X, double \*R, double \*P)



- *Evaluate the objective Z function.*
- double **eval\_Z\_** ()
- double **eval\_dZ\_** (double g, double \*R, double \*P, int i, int j)
- *Evaluate the change in Z.*
- double **eval\_Z\_trial\_** (int i, int j, double gamma)
- *Evaluate the trial Z value.*
- void **form\_X0\_** ()
- *Create identity matrix.*
- void **form\_X\_** (int i, int j, double gamma)
- *Form unitary matrix X (store in buffer Xnew\_)*
- void **form\_next\_X\_** (const std::string &opt)
- *Form the next unitary matrix X.*
- **ABCD get\_ABCD\_** (int i, int j)
- *Retrieve ABCD parameters for root search.*
- void **find\_roots\_boyd\_** (const **ABCD** &abcd)
- *Solve for all roots of equation  $A*\sin(g) + B*\cos(g) + C*\sin(2*g) + D*\cos(2*g) = 0$  -> implements Boyd's method.*
- double **find\_root\_halley\_** (double x0, const **ABCD** &abcd)
- *Solve for root of equation  $A*\sin(g) + B*\cos(g) + C*\sin(2*g) + D*\cos(2*g) = 0$  -> implements Halley's method.*
- double **find\_gamma\_** (const **ABCD** &abcd, int i, int j, const std::string &opt)
- *Compute gamma from roots of base equations.*
- bool **lt\_** (double a, double b)
- *less-than function*
- bool **gt\_** (double a, double b)
- *greater-than function*
- double **func\_0\_** (double g, const **ABCD** &abcd)
- *Function  $f(\text{gamma}) = d(dZ)/d\text{gamma}$ .*
- double **func\_1\_** (double g, const **ABCD** &abcd)
- *Gradient of  $f(\text{gamma})$*
- double **func\_2\_** (double g, const **ABCD** &abcd)
- *Hessian of  $f(\text{gamma})$  - used only for Halley method (not implemented since Boyd method is more suitable here)*
- std::shared\_ptr< psi::Matrix > **psi\_X\_** ()
- *Form the Psi4 matrix with the transformation matrix.*

## Protected Attributes

- const int **n\_**
- *Dimension of the problem.*
- const double **conv\_**
- *Convergence.*
- const int **maxiter\_**
- *Maximum number of iterations.*
- const bool **verbose\_**
- *Verbose mode.*
- double \* **R\_**
- *R matrix.*
- double \* **P\_**
- *P vector.*
- double \* **R0\_**
- *Reference R matrix.*
- double \* **P0\_**

- *Reference P vector.*
- double \* [X\\_](#)  
*X Matrix (accumulated solution)*
- double \* [W\\_](#)  
*Work place.*
- double \* [Xold\\_](#)  
*Temporary X matrix.*
- double \* [Xnew\\_](#)  
*Temporary X matrix.*
- int [niter\\_](#)  
*Current number of iterations.*
- double [S\\_](#)[4]  
*Current solutions.*
- double [Zinit\\_](#)  
*Initial Z value.*
- double [Zold\\_](#)  
*Old Z value.*
- double [Znew\\_](#)  
*New Z value.*
- double [conv\\_current\\_](#)  
*Current convergence.*
- bool [success\\_](#)  
*Status of optimization.*

### 17.64.1 Detailed Description

Find the optimim unitary matrix of quadratic matrix equation.

The objective function of the orthogonal matrix  $\mathbf{X}$

$$Z(\mathbf{X}) \equiv \sum_{ijkl} X_{ij} X_{kl} R_{jl} - \sum_{ij} X_{ij} P_j$$

is optimized by using the Jacobi iteration algorithm. In the above equation,  $\mathbf{R}$  is a square, general real matrix of size  $N \times N$  whereas  $\mathbf{P}$  is a real vector of length  $N$ .

**Algorithm.**

Optimization of  $\mathbf{X}$  is factorized into a sequence of 2-dimensional rotations with one real parameter  $\gamma$ :

$$\mathbf{X}^{\text{New}} = \mathbf{U}(\gamma) \cdot \mathbf{X}^{\text{Old}}$$

where

$$\mathbf{U}(\gamma) \equiv \begin{pmatrix} \ddots & & & & \\ & \cos(\gamma) & \cdots & \sin(\gamma) & \\ & \vdots & \ddots & \vdots & \\ & -\sin(\gamma) & \cdots & \cos(\gamma) & \\ & & & & \ddots \end{pmatrix}$$

is the Jacobi transformation matrix constructed for the  $I$ th and  $J$ th element from the entire  $N$ -dimensional set. For the sake of algorithmic simplicity, every iteration after  $\mathbf{U}(\gamma)$  has been formed,  $\mathbf{X}^{\text{Old}}$  is for a while assumed to be an identity matrix and the  $\mathbf{R}$  matrix and  $\mathbf{P}$  vector are transformed according to the following formulae

$$\begin{aligned} \mathbf{R} &\rightarrow \mathbf{U}\mathbf{R}\mathbf{U}^T \\ \mathbf{P} &\rightarrow \mathbf{U}\mathbf{P} \end{aligned}$$

The full transformation matrix is accumulated in the memory buffer until convergence.

In each iteration, the optimum angle  $\gamma$  is found as follows: First, the roots of the finite Fourier series

$$A \sin(\gamma) + B \cos(\gamma) + C \sin(2\gamma) + D \cos(2\gamma) = 0$$

are found. In the above equations, the expansion coefficients are given as

$$\begin{aligned} A &= P_I + P_J - \sum_{k \neq I, J} (R_{Ik} + R_{Jk} + R_{kI} + R_{kJ}) \\ B &= P_I - P_J - \sum_{k \neq I, J} (R_{Ik} - R_{Jk} + R_{kI} - R_{kJ}) \\ C &= -2(R_{IJ} + R_{JI}) \\ D &= -2(R_{II} - R_{JJ}) \end{aligned}$$

and  $I, J$  are the chosen indices in the Jacobi iteration subspace. The roots are evaluated by applying the Boyd's method[1], in which they are given as

$$\gamma_n = \Re[-i \ln(\lambda_n)]$$

where  $\lambda_n$  is an eigenvalue of the following 4 by 4 complex matrix:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{D+iC}{D-iC} & -\frac{B+iC}{D-iC} & 0 & -\frac{B-iC}{D-iC} \end{pmatrix}$$

Once the four roots of the Fourier series equation are found, one solution out of four is chosen which satisfies the global optimum condition, i.e., the largest increase/decrease in the objective function given by

$$\delta Z = A(1 - \cos(\gamma)) + B \sin(\gamma) + C \sin^2(\gamma) + \frac{D}{2} \sin(2\gamma)$$

The discrimination between the minima/maxima is performed based on the evaluation of the Hessian of  $Z$  with respect to  $\gamma$ ,

$$\frac{\partial^2 Z}{\partial \gamma^2} = A \cos(\gamma) - B \sin(\gamma) + 2C \cos(2\gamma) - 2D \sin(2\gamma)$$

All the  $N(N-1)/2$  unique pairs of molecular orbitals are checked and the optimal set of  $\gamma, I, J$  is chosen to construct  $\mathbf{X}^{\text{New}}$ .

#### References:

[1] Boyd, J.P.; J. Eng. Math. (2006) 56, pp. 203-219

### 17.64.2 Constructor & Destructor Documentation

**17.64.2.1** oepdev::UnitaryOptimizer::UnitaryOptimizer ( double \*  $R$ , double \*  $P$ , int  $n$ , double  $conv = 1.0e-6$ , int  $maxiter = 100$ , bool  $verbose = true$  )

Create from  $R$  and  $P$  matrices and optimization options.

#### Parameters

$R$	- $\mathbf{R}$ matrix
$P$	- $\mathbf{P}$ vector
$n$	- dimensionality of the problem ( $N$ )

<i>conv</i>	- convergence in the $Z$ function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

17.64.2.2 `oepdev::UnitaryOptimizer::UnitaryOptimizer ( std::shared_ptr< psi::Matrix > R, std::shared_ptr< psi::Vector > P, double conv = 1.0e-6, int maxiter = 100, bool verbose = true )`

Create from  $R$  and  $P$  matrices and optimization options.

Parameters

<i>R</i>	- $\mathbf{R}$ matrix
<i>P</i>	- $\mathbf{P}$ vector
<i>conv</i>	- convergence in the $Z$ function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

17.64.2.3 `oepdev::UnitaryOptimizer::UnitaryOptimizer ( int n, double conv, int maxiter, bool verbose )` [protected]

Initialize the basic memory.

Parameters

<i>n</i>	- dimensionality of the problem ( $N$ )
<i>conv</i>	- convergence in the $Z$ function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

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

- [oepdev/libutil/unitary\\_optimizer.h](#)
- [oepdev/libutil/unitary\\_optimizer.cc](#)

## 17.65 oepdev::UnitaryOptimizer\_4\_2 Class Reference

Find the optimim unitary matrix for quartic-quadratic matrix equation with trace.

```
#include <unitary_optimizer.h>
```

### Public Member Functions

- [UnitaryOptimizer\\_4\\_2](#) (double \* $R$ , double \* $P$ , int  $n$ , double  $conv=1.0e-6$ , int  $maxiter=100$ , bool  $verbose=true$ )  
Create from  $R$  and  $P$  matrices and optimization options.
- [~UnitaryOptimizer\\_4\\_2](#) ()  
Clear memory.
- bool [maximize](#) ()  
Run the minimization.
- bool [minimize](#) ()  
Run the maximization.
- std::shared\_ptr< psi::Matrix > [X](#) ()  
Get the unitary matrix (solution)
- double \* [get\\_X](#) () const  
Get the unitary matrix (pointer to solution)

- double [Z](#) ()  
*Get the actual value of Z function.*
- bool [success](#) () const  
*Get the status of the optimization.*

## Protected Member Functions

- [UnitaryOptimizer\\_4\\_2](#) (int n, double conv, int maxiter, bool verbose)  
*Initialize the basic memory.*
- void [common\\_init](#) ()  
*Prepare the optimizer.*
- void [run](#) (const std::string &opt)  
*Run the optimization (intermediate interface)*
- void [optimize](#) (const std::string &opt)  
*Run the optimization (inner interface)*
- void [refresh](#) ()  
*Restore the initial state of the optimizer.*
- void [update\\_conv](#) ()  
*Update the convergence.*
- void [update\\_iter](#) ()  
*Update the iterates.*
- void [update\\_Z](#) ()  
*Update Z value.*
- void [update\\_RP](#) ()  
*Uptade R and P matrices.*
- void [update\\_X](#) ()  
*Update the solution matrix X.*
- double [eval\\_Z](#) (double \*X, double \*R, double \*P)  
*Evaluate the objective Z function.*
- double [eval\\_Z](#) ()
- double [eval\\_dZ](#) (double g, double \*R, double \*P, int I, int J)  
*Evaluate the change in Z.*
- double [eval\\_Z\\_trial](#) (int I, int J, double gamma)  
*Evaluate the trial Z value.*
- void [form\\_X0](#) ()  
*Create identity matrix.*
- void [form\\_X](#) (int I, int J, double gamma)  
*Form unitary matrix X (store in buffer Xnew\_)*
- void [form\\_next\\_X](#) (const std::string &opt)  
*Form the next unitary matrix X.*
- [Fourier9](#) [get\\_fourier](#) (int I, int J)  
*Retrieve ABCD parameters for root search.*
- void [find\\_roots\\_boyd](#) (const [Fourier9](#) &abcd)  
*Solve for all roots of equation  $A*\sin(g) + B*\cos(g) + C*\sin(2*g) + D*\cos(2*g) = 0$  -> implements Boyd's method.*
- double [find\\_root\\_halley](#) (double x0, const [Fourier9](#) &abcd)  
*Solve for root of equation  $A*\sin(g) + B*\cos(g) + C*\sin(2*g) + D*\cos(2*g) = 0$  -> implements Halley's method.*
- double [find\\_gamma](#) (const [Fourier9](#) &abcd, int i, int j, const std::string &opt)  
*Compute gamma from roots of base equations.*
- bool [lt](#) (double a, double b)  
*less-than function*

- bool [gt\\_](#) (double a, double b)  
*greater-than function*
- std::shared\_ptr< psi::Matrix > [psi\\_X\\_](#) ()  
*Form the Psi4 matrix with the transformation matrix.*

## Protected Attributes

- const int [n\\_](#)  
*Dimension of the problem.*
- const double [conv\\_](#)  
*Convergence.*
- const int [maxiter\\_](#)  
*Maximum number of iterations.*
- const bool [verbose\\_](#)  
*Verbose mode.*
- double \* [R\\_](#)  
*R tensor.*
- double \* [P\\_](#)  
*P tensor.*
- double \* [R0\\_](#)  
*Reference R tensor.*
- double \* [P0\\_](#)  
*Reference P tensor.*
- double \* [X\\_](#)  
*X Matrix (accumulated solution)*
- double \* [W\\_](#)  
*Work place.*
- double \* [Xold\\_](#)  
*Temporary X matrix.*
- double \* [Xnew\\_](#)  
*Temporary X matrix.*
- int [niter\\_](#)  
*Current number of iterations.*
- double [S\\_](#) [8]  
*Current solutions.*
- double [Zinit\\_](#)  
*Initial Z value.*
- double [Zold\\_](#)  
*Old Z value.*
- double [Znew\\_](#)  
*New Z value.*
- double [conv\\_current\\_](#)  
*Current convergence.*
- bool [success\\_](#)  
*Status of optimization.*

### 17.65.1 Detailed Description

Find the optimum unitary matrix for quartic-quadratic matrix equation with trace.

The objective function of the orthogonal matrix  $\mathbf{X}$

$$Z(\mathbf{X}) \equiv \sum_{ijklmn} X_{ki} X_{lj} X_{mi} X_{nj} R_{ijklmn} + \sum_{ijk} X_{ji} X_{ki} P_{ijk}$$

is optimized by using the Jacobi iteration algorithm. In the above equation,  $\mathbf{R}$  is a general real sixth-rank tensor of size  $N^6$  whereas  $\mathbf{P}$  is a general real third-rank tensor of size  $N^3$ .

**Algorithm.**

Optimization of  $\mathbf{X}$  is factorized into a sequence of 2-dimensional rotations with one real parameter  $\gamma$ :

$$\mathbf{X}^{\text{New}} = \mathbf{X}^{\text{Old}} \cdot \mathbf{U}(\gamma)$$

where

$$\mathbf{U}(\gamma) \equiv \begin{pmatrix} \ddots & & & & & & & \\ & \cos(\gamma) & \cdots & \sin(\gamma) & & & & \\ & \vdots & \ddots & \vdots & & & & \\ & -\sin(\gamma) & \cdots & \cos(\gamma) & & & & \\ & & & & \ddots & & & \end{pmatrix}$$

is the Jacobi transformation matrix constructed for the  $I$ th and  $J$ th element from the entire  $N$ -dimensional set. For the sake of algorithmic simplicity, every iteration after  $\mathbf{U}(\gamma)$  has been formed,  $\mathbf{X}^{\text{Old}}$  is for a while assumed to be an identity matrix and the  $\mathbf{R}$  as well as  $\mathbf{P}$  tensors are transformed according to the following formulae

$$R_{ijklmn} \rightarrow \sum_{k'l'm'n'} R_{ijkl'm'n'} X_{k'k} X_{l'l} X_{m'm} X_{n'n}$$

$$P_{ijk} \rightarrow \sum_{j'k'} P_{ij'k'} X_{j'j} X_{k'k}$$

The full transformation matrix is accumulated in the memory buffer until convergence.

In each iteration, the optimum angle  $\gamma$  is found as follows: First, the roots of the finite Fourier series

$$a_0 + \sum_{p=1}^4 \{a_p \cos(px) + b_p \sin(px)\} = 0$$

are found. In the above equations, the expansion coefficients are calculated analytically as a function of  $I, J$  - the chosen indices in the Jacobi iteration subspace. The roots are evaluated by applying the Boyd's method[1], in which they are given as

$$\gamma_n = \Re[-i \ln(\lambda_n)]$$

where  $\lambda_n$  is an eigenvalue of the following 8 by 8 complex matrix:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -\frac{a_4+ib_4}{a_4-ib_4} & -\frac{a_3+ib_3}{a_4-ib_4} & -\frac{a_2+ib_2}{a_4-ib_4} & -\frac{a_1+ib_1}{a_4-ib_4} & -\frac{2a_0}{a_4-ib_4} & -\frac{a_1-ib_1}{a_4-ib_4} & -\frac{a_2-ib_2}{a_4-ib_4} & -\frac{a_3-ib_3}{a_4-ib_4} \end{pmatrix}$$

Once the eight roots of the Fourier series equation are found, one solution out of eight is chosen which satisfies the global optimum condition, i.e., the largest increase/decrease in the objective function given by

$$\delta Z = Z(\mathbf{U}(\gamma)) - Z(\mathbf{1})$$

The Hessian is not computed. All the  $N(N-1)/2$  unique pairs of molecular orbitals are checked and the optimal set of  $\gamma, I, J$  is chosen to construct  $\mathbf{X}^{\text{New}}$ .

## References:

[1] Boyd, J.P.; J. Eng. Math. (2006) 56, pp. 203-219

## 17.65.2 Constructor & Destructor Documentation

17.65.2.1 `oepdev::UnitaryOptimizer_4_2::UnitaryOptimizer_4_2 ( double * R, double * P, int n, double conv = 1.0e-6, int maxiter = 100, bool verbose = true )`

Create from R and P matrices and optimization options.

## Parameters

<i>R</i>	- <b>R</b> tensor (flattened row-wise)
<i>P</i>	- <b>P</b> tensor (flattened row-wise)
<i>n</i>	- dimensionality of the problem ( <i>N</i> )
<i>conv</i>	- convergence in the <i>Z</i> function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

17.65.2.2 `oepdev::UnitaryOptimizer_4_2::UnitaryOptimizer_4_2 ( int n, double conv, int maxiter, bool verbose )`  
[protected]

Initialize the basic memory.

## Parameters

<i>n</i>	- dimensionality of the problem ( <i>N</i> )
<i>conv</i>	- convergence in the <i>Z</i> function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

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

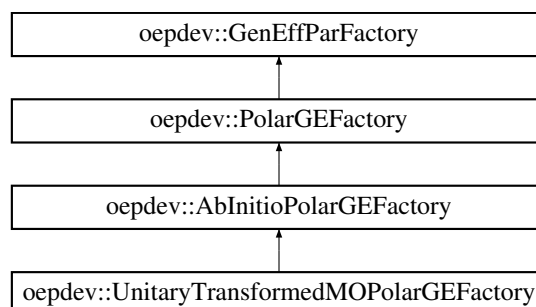
- [oepdev/libutil/unitary\\_optimizer.h](#)
- [oepdev/libutil/unitary\\_optimizer.cc](#)

## 17.66 oepdev::UnitaryTransformedMOPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Scaling of MO Space.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::UnitaryTransformedMOPolarGEFactory:





## Public Member Functions

- [UnitaryTransformedMOPolarGEFactory](#) (std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)  
*Construct from CPHF object and Psi4 options.*
- virtual [~UnitaryTransformedMOPolarGEFactory](#) ()  
*Destruct.*
- std::shared\_ptr< [GenEffPar](#) > [compute](#) (void)  
*Perform Least-Squares Fit.*

## Additional Inherited Members

### 17.66.1 Detailed Description

Polarization GEFP Factory with Least-Squares Scaling of MO Space.

Implements creation of the density matrix susceptibility tensors for which  $\mathbf{X} \neq \mathbf{1}$ . Guarantees the idempotency of the density matrix up to first-order in LCAO-MO variation.

#### Note

This method does not give better results than the  $X=1$  method and is extremely time and memory consuming. Therefore, it is placed here only for future reference about solving unitary optimization problem in case it occurs.

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

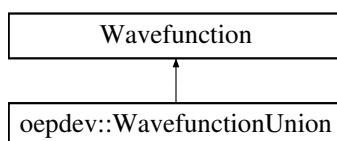
- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp\_polar\_abinitio.cc

## 17.67 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.*
- SharedBasisSet [l\\_intermediate](#) (int n) const  
*Get the intermediate 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< SharedBasisSet > [l\\_intermediate\\_](#)  
*List of intermediate 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\\_alpha\\_](#)

- 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.*

### 17.67.1 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)
- `frzvpi_` (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.

## 17.67.2 Constructor & Destructor Documentation

### 17.67.2.1 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

## 17.67.3 Member Function Documentation

### 17.67.3.1 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

### 17.67.3.2 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

## Chapter 18

# File Documentation

### 18.1 include/oepdev\_files.h File Reference

#### Macros

- `#define OEPDEV_USE_PSI4_DIIS_MANAGER 0`  
*Use DIIS from Psi4 (1) or OEPDev (0)?*
- `#define OEPDEV_MAX_AM 8`  
*L\_max.*
- `#define OEPDEV_N_MAX_AM 17`  
*2L\_max+1*
- `#define OEPDEV_CRIT_ERI 1e-9`  
*ERI criterion for E12, E34, E123 and lambda\*EXY coefficients.*
- `#define OEPDEV_SIZE_BUFFER_R 250563`  
*Size of R buffer (OEPDEV\_N\_MAX\_AM\*OEPDEV\_N\_MAX\_AM\*OEPDEV\_N\_MAX\_AM\*OEPDEV\_N\_MAX\_AM\*3)*
- `#define OEPDEV_SIZE_BUFFER_D2 3264`  
*Size of D2 buffer (3\*(OEPDEV\_MAX\_AM+1)\*(OEPDEV\_MAX\_AM+1)\*OEPDEV\_N\_MAX\_AM)*

### 18.2 include/oepdev\_options.h File Reference

#### Namespaces

- `psi`  
*Psi4 package namespace.*

#### Functions

- `int psi::read_options (std::string name, Options &options)`  
*Options for the OEPDev plugin.*

### 18.3 main.cc File Reference

```
#include <string>
#include "include/oepdev_files.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libmints/wavefunction.h"
#include "include/oepdev_options.h"
#include "oepdev/liboep/oep.h"
#include "oepdev/libgefp/gefp.h"
#include "oepdev/libsolver/solver.h"
#include "oepdev/libtest/test.h"
```

#### Namespaces

- [psi](#)

*Psi4 package namespace.*

#### Typedefs

- using **SharedWavefunction** = std::shared\_ptr< [psi::Wavefunction](#) >
- using **SharedUnion** = std::shared\_ptr< [oepdev::WavefunctionUnion](#) >
- using **SharedOEPotential** = std::shared\_ptr< [oepdev::OEPotential](#) >
- using **SharedGEFFPFactory** = std::shared\_ptr< [oepdev::GenEffParFactory](#) >
- using **SharedGEFFParameters** = std::shared\_ptr< [oepdev::GenEffPar](#) >

#### Functions

- [SharedWavefunction](#) [psi::oepdev](#) ([SharedWavefunction](#) ref\_wfn, Options &options)

*Main routine of the OEPDev plugin.*

### 18.4 oepdev/lib3d/dmtp.h File Reference

```
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/molecule.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/vector.h"
```

#### Classes

- class [oepdev::MultipoleConvergence](#)

*Multipole Convergence.*

- class [oepdev::DMTPole](#)

*Distributed Multipole Analysis Container and Computer. Abstract Base.*

- class [oepdev::CAMM](#)

*Cumulative Atomic Multipole Moments.*



## Namespaces

- [psi](#)  
*Psi4 package namespace.*
- [oepdev](#)  
*OEPEDev module namespace.*

## Typedefs

- using **psi::SharedBasisSet** = std::shared\_ptr< BasisSet >

## 18.5 oepdev/lib3d/esp.h File Reference

```
#include "space3d.h"
```

## Classes

- class [oepdev::ESPSolver](#)  
*Charges from Electrostatic Potential (ESP). A solver-type class.*

## Namespaces

- [oepdev](#)  
*OEPEDev module namespace.*

## Typedefs

- using **oepdev::SharedField3D** = std::shared\_ptr< [oepdev::Field3D](#) >

## 18.6 oepdev/libgefp/gefp.h File Reference

```
#include <vector>
#include <string>
#include <random>
#include <cmath>
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/vector3.h"
#include "../liboep/oep.h"
#include "../libutil/util.h"
#include "../libutil/cphf.h"
#include "../libutil/scf_perturb.h"
```

## Classes

- class [oepdev::GenEffPar](#)  
*Generalized Effective Fragment Parameters. Container Class.*
- class [oepdev::GenEffFrag](#)  
*Generalized Effective Fragment. Container Class.*
- class [oepdev::GenEffParFactory](#)  
*Generalized Effective Fragment Factory. Abstract Base.*
- class [oepdev::PolarGEFactory](#)  
*Polarization GEFP Factory. Abstract Base.*
- class [oepdev::AbInitioPolarGEFactory](#)  
*Polarization GEFP Factory from First Principles. Hartree-Fock Approximation.*
- class [oepdev::FFAbInitioPolarGEFactory](#)  
*Polarization GEFP Factory from First Principles: Finite-Difference Model. Arbitrary level of theory.*
- class [oepdev::GeneralizedPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- struct [oepdev::GeneralizedPolarGEFactory::StatisticalSet](#)  
*A structure to handle statistical data.*
- class [oepdev::UniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::NonUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::LinearUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::QuadraticUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::LinearNonUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::QuadraticNonUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::LinearGradientNonUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::UnitaryTransformedMOPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Scaling of MO Space.*

## Namespaces

- [oepdev](#)  
*OEPEDev module namespace.*

## 18.7 oepdev/libints/eri.h File Reference

```
#include "psi4/libpsi4util/exception.h"
#include "psi4/libmints/integral.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/fjt.h"
#include "../libpsi/integral.h"
#include "recurr.h"
```

## Classes

- class [oepdev::TwoElectronInt](#)  
*General Two Electron Integral.*
- class [oepdev::ERI\\_1\\_1](#)  
*2-centre ERI of the form  $\langle a|O(2)|b \rangle$  where  $O(2) = 1/r12$ .*
- class [oepdev::ERI\\_2\\_2](#)  
*4-centre ERI of the form  $\langle ab|O(2)|cd \rangle$  where  $O(2) = 1/r12$ .*
- class [oepdev::ERI\\_3\\_1](#)  
*4-centre ERI of the form  $\langle abc|O(2)|d \rangle$  where  $O(2) = 1/r12$ .*

## Namespaces

- [oepdev](#)  
*OEPEv module namespace.*

## 18.8 oepdev/libints/recurr.h File Reference

### Namespaces

- [oepdev](#)  
*OEPEv module namespace.*

### Macros

- `#define D1_INDEX(x, i, n) ((81*(x))+(9*(i))+(n))`  
*Get the index of McMurchie-Davidson-Hermite D1 coefficient stored in the `mdh_buffer_`, that is attributed to the x Cartesian coordinate from angular momentum i of function 1, and the Hermite index n.*
- `#define D2_INDEX(x, i, j, n) ((1377*(x))+(153*(i))+(17*(j))+(n))`  
*Get the index of McMurchie-Davidson-Hermite D2 coefficient stored in the `mdh_buffer_`, that is attributed to the x Cartesian coordinate from angular momenta i, j of function 1 and 2, and the Hermite index n.*
- `#define D3_INDEX(x, i, j, k, n) ((18225*(x))+(2025*(i))+(225*(j))+(25*(k))+(n))`  
*Get the index of McMurchie-Davidson-Hermite D3 coefficient stored in the `mdh_buffer_`, that is attributed to the x Cartesian coordinate from angular momenta i, j and k of function 1, 2 and 3, and the Hermite index n.*
- `#define R_INDEX(n, l, m, j) ((14739*(n))+(867*(l))+(51*(m))+(j))`  
*Get the index of McMurchie-Davidson R coefficient stored in the `mdh_buffer_R_` from angular momenta n, l and m and the Boys index j.*

### Functions

- double [oepdev::d\\_N\\_n1\\_n2](#) (int N, int n1, int n2, double PA, double PB, double aP)  
*Compute McMurchie-Davidson-Hermite (MDH) coefficient for binomial expansion.*
- void [oepdev::make\\_mdh\\_D1\\_coeff](#) (int n1, double aPd, double \*buffer)  
*Compute the McMurchie-Davidson-Hermite coefficients for monomial expansion.*
- void [oepdev::make\\_mdh\\_D2\\_coeff](#) (int n1, int n2, double aPd, double \*PA, double \*PB, double \*buffer)  
*Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion.*
- void [oepdev::make\\_mdh\\_D3\\_coeff](#) (int n1, int n2, int n3, double aPd, double \*PA, double \*PB, double \*PC, double \*buffer)  
*Compute the McMurchie-Davidson-Hermite coefficients for trinomial expansion.*
- void [oepdev::make\\_mdh\\_D2\\_coeff\\_explicit\\_recursion](#) (int n1, int n2, double aP, double \*PA, double \*PB, double \*buffer)

Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion by explicit recursion. This function makes the same changes to buffers as `oepdev::make_mdh_D2_coeff`, but implements it through explicit recursion by calls to `oepdev::d_N_n1_n2`. Therefore, it is slightly slower. Here for debugging purposes.

- void `oepdev::make_mdh_R_coeff` (int N, int L, int M, double alpha, double a, double b, double c, double \*F, double \*buffer)

Compute the McMurchie-Davidson R coefficients.

## 18.9 oepdev/liboep/oep.h File Reference

```
#include <cstdio>
#include <string>
#include <vector>
#include <map>
#include "psi4/libparallel/parallel.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/vector.h"
#include "../libpsi/integral.h"
#include "../libpsi/potential.h"
#include "../lib3d/space3d.h"
#include "../lib3d/dmtp.h"
```

### Classes

- struct `oepdev::OEType`  
Container to handle the type of One-Electron Potentials.
- class `oepdev::OEPotential`  
Generalized One-Electron Potential: Abstract base.
- class `oepdev::ElectrostaticEnergyOEPotential`  
Generalized One-Electron Potential for Electrostatic Energy.
- class `oepdev::RepulsionEnergyOEPotential`  
Generalized One-Electron Potential for Pauli Repulsion Energy.
- class `oepdev::ChargeTransferEnergyOEPotential`  
Generalized One-Electron Potential for Charge-Transfer Interaction Energy.
- class `oepdev::EETCouplingOEPotential`  
Generalized One-Electron Potential for EET coupling calculations.

### Namespaces

- `oepdev`  
OEPDev module namespace.

### Typedefs

- using `oepdev::SharedWavefunction` = `std::shared_ptr< Wavefunction >`
- using `oepdev::SharedBasisSet` = `std::shared_ptr< BasisSet >`
- using `oepdev::SharedMatrix` = `std::shared_ptr< Matrix >`
- using `oepdev::SharedVector` = `std::shared_ptr< Vector >`
- using `oepdev::SharedDMTPole` = `std::shared_ptr< DMTPole >`

## 18.10 oepdev/liboep/oep\_gdf.h File Reference

```
#include <cstdio>
#include <string>
#include "psi4/liboptions/liboptions.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/matrix.h"
#include "../libpsi/integral.h"
```

### Classes

- class [oepdev::GeneralizedDensityFit](#)  
*Generalized Density Fitting Scheme. Abstract Base.*
- class [oepdev::SingleGeneralizedDensityFit](#)  
*Generalized Density Fitting Scheme - Single Fit.*
- class [oepdev::DoubleGeneralizedDensityFit](#)  
*Generalized Density Fitting Scheme - Double Fit.*

### Namespaces

- [oepdev](#)  
*OEPPDev module namespace.*

## 18.11 oepdev/libpsi/integral.h File Reference

```
#include "psi4/libmints/integral.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/matrix.h"
```

### Classes

- class [oepdev::TwoBodyAOInt](#)
- class [oepdev::IntegralFactory](#)  
*Extended [IntegralFactory](#) for computing integrals.*

### Namespaces

- [oepdev](#)  
*OEPPDev module namespace.*

## 18.12 oepdev/libpsi/potential.h File Reference

```
#include <vector>
#include "psi4/psi4-dec.h"
#include "psi4/libparallel/parallel.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libmints/typedefs.h"
#include "psi4/libmints/onebody.h"
#include "psi4/libmints/potential.h"
#include "psi4/libmints/sointegral_onebody.h"
#include "psi4/libmints/osrecur.h"
```

### Classes

- class [oepdev::PotentialInt](#)  
*Computes potential integrals.*

### Namespaces

- [oepdev](#)  
*OEPEv module namespace.*

## 18.13 oepdev/libsolver/solver.h File Reference

```
#include <cstdio>
#include <string>
#include <map>
#include "psi4/psi4-dec.h"
#include "psi4/libparallel/parallel.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libmints/potential.h"
#include "psi4/libmints/integral.h"
#include "../libutil/wavefunction_union.h"
#include "../libutil/integrals_iter.h"
#include "../libpsi/integral.h"
#include "../liboep/oep.h"
```

### Classes

- class [oepdev::OEPEvSolver](#)  
*Solver of properties of molecular aggregates. Abstract base.*
- class [oepdev::ElectrostaticEnergySolver](#)  
*Compute the Coulombic interaction energy between unperturbed wavefunctions.*
- class [oepdev::RepulsionEnergySolver](#)  
*Compute the Pauli-Repulsion interaction energy between unperturbed wavefunctions.*
- class [oepdev::ChargeTransferEnergySolver](#)  
*Compute the Charge-Transfer interaction energy between unperturbed wavefunctions.*

## Namespaces

- [oepdev](#)  
*OEPCDev module namespace.*

## Typedefs

- using **oepdev::SharedWavefunctionUnion** = std::shared\_ptr< WavefunctionUnion >
- using **oepdev::SharedOEPotential** = std::shared\_ptr< OEPotential >

## 18.14 oepdev/libtest/test.h File Reference

```
#include <vector>
#include "psi4/psi4-dec.h"
#include "psi4/libparallel/parallel.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libmints/integral.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libqt/qt.h"
#include "../libpsi/integral.h"
#include "../libutil/integrals_iter.h"
```

## Classes

- class [oepdev::test::Test](#)  
*Manages test routines.*

## Namespaces

- [oepdev](#)  
*OEPCDev module namespace.*

## 18.15 oepdev/libutil/diis.h File Reference

```
#include <cstdio>
#include <string>
#include <vector>
#include "psi4/libparallel/parallel.h"
#include "psi4/libciomr/libciomr.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libqt/qt.h"
```

## Classes

- class [oepdev::DIISManager](#)  
*DIIS manager.*

## Namespaces

- [oepdev](#)

*OEPPDev module namespace.*

## 18.16 oepdev/libutil/integrals\_iter.h File Reference

```
#include <cstdio>
#include "psi4/libparallel/parallel.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/integral.h"
#include "../libpsi/integral.h"
```

## Classes

- class [oepdev::ShellCombinationsIterator](#)  
*Iterator for Shell Combinations. Abstract Base.*
- class [oepdev::AOIntegralsIterator](#)  
*Iterator for AO Integrals. Abstract Base.*
- class [oepdev::AllAOShellCombinationsIterator\\_4](#)  
*Loop over all possible ERI shells in a shell quartet.*
- class [oepdev::AllAOShellCombinationsIterator\\_2](#)  
*Loop over all possible ERI shells in a shell doublet.*
- class [oepdev::AllAOIntegralsIterator\\_4](#)  
*Loop over all possible ERI within a particular shell quartet.*
- class [oepdev::AllAOIntegralsIterator\\_2](#)  
*Loop over all possible ERI within a particular shell doublet.*

## Namespaces

- [oepdev](#)

*OEPPDev module namespace.*

## Typedefs

- using [oepdev::SharedIntegralFactory](#) = std::shared\_ptr< IntegralFactory >
- using [oepdev::SharedTwoBodyAOInt](#) = std::shared\_ptr< TwoBodyAOInt >
- using [oepdev::SharedShellsIterator](#) = std::shared\_ptr< ShellCombinationsIterator >  
*Iterator over shells as shared pointer.*
- using [oepdev::SharedAOIntsIterator](#) = std::shared\_ptr< AOIntegralsIterator >  
*Iterator over AO integrals as shared pointer.*

## 18.17 oepdev/libutil/scf\_perturb.h File Reference

```
#include "psi4/libmints/vector.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libscf_solver/rhf.h"
```



## Classes

- struct [oepdev::PerturbCharges](#)  
*Structure to hold perturbing charges.*
- class [oepdev::RHFPerturbed](#)  
*RHF theory under electrostatic perturbation.*

## Namespaces

- [oepdev](#)  
*OEPEv module namespace.*

## 18.18 oepdev/libutil/unitary\_optimizer.h File Reference

```
#include <string>
#include <complex>
#include "psi4/libmints/vector.h"
#include "psi4/libmints/matrix.h"
```

## Classes

- struct [oepdev::ABCD](#)  
*Simple structure to hold the Fourier series expansion coefficients.*
- struct [oepdev::Fourier9](#)  
*Simple structure to hold the Fourier series expansion coefficients for N=4.*
- class [oepdev::UnitaryOptimizer](#)  
*Find the optimim unitary matrix of quadratic matrix equation.*
- class [oepdev::UnitaryOptimizer\\_4\\_2](#)  
*Find the optimim unitary matrix for quartic-quadratic matrix equation with trace.*

## Namespaces

- [oepdev](#)  
*OEPEv module namespace.*

## Macros

- `#define IDX(i, j, n) ((n)*(i)+(j))`
- `#define IDX3(i, j, k) (n2_*(i)+n_*(j)+(k))`
- `#define IDX6(i, j, k, l, m, n) (n5_*(i)+n4_*(j)+n3_*(k)+n2_*(l)+n_*(m)+(n))`

## Functions

- `constexpr std::complex< double > oepdev::operator""_i (unsigned long long d)`
- `constexpr std::complex< double > oepdev::operator""_i (long double d)`

## 18.19 oepdev/libutil/util.h File Reference

```
#include <cstdio>
#include <string>
#include <cmath>
#include <map>
#include "psi4/psi4-dec.h"
#include "psi4/libparallel/parallel.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libciomr/libciomr.h"
#include "psi4/libpsio/psio.hpp"
#include "psi4/libiwl/iwl.h"
#include "psi4/libqt/qt.h"
#include "psi4/libmints/molecule.h"
#include "psi4/libmints/writer.h"
#include "psi4/libmints/writer_file_prefix.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/oeprop.h"
#include "psi4/libmints/local.h"
#include "psi4/libfunctional/superfunctional.h"
#include "psi4/libtrans/mospace.h"
#include "psi4/libtrans/integraltransform.h"
#include "psi4/libscf_solver/rhf.h"
#include "psi4/libdpd/dpd.h"
```

### Namespaces

- [oepdev](#)

*OEPDev module namespace.*

### Typedefs

- using [oepdev::SharedMolecule](#) = std::shared\_ptr< Molecule >
- using [oepdev::SharedSuperFunctional](#) = std::shared\_ptr< SuperFunctional >
- using [oepdev::SharedMOSpace](#) = std::shared\_ptr< MOSpace >
- using [oepdev::SharedMOSpaceVector](#) = std::vector< std::shared\_ptr< MOSpace >>
- using [oepdev::SharedIntegralTransform](#) = std::shared\_ptr< IntegralTransform >
- using [oepdev::SharedLocalizer](#) = std::shared\_ptr< Localizer >

### Functions

- void [oepdev::preamble](#) (void)  
*Print preamble for module OEPDEV.*
- template<typename... Args>  
std::string [oepdev::string\\_sprintf](#) (const char \*format, Args...args)  
*Format string output. Example: std::string text = oepdev::string\_sprintf("Test %3d, %13.5f", 5, -10.5425);.*
- std::shared\_ptr< SuperFunctional > [oepdev::create\\_superfunctional](#) (std::string name, Options &options)  
*Set up DFT functional.*

- `std::shared_ptr< Molecule > oepdev::extract_monomer` (`std::shared_ptr< const Molecule > molecule_dimer, int id`)  
*Extract molecule from dimer.*
- `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.*
- `double oepdev::average_moment` (`std::shared_ptr< psi::Vector > moment`)  
*Compute the scalar magnitude of multipole moment.*

## 18.20 oepdev/libutil/wavefunction\_union.h File Reference

```
#include <cstdio>
#include <string>
#include <map>
#include "psi4/psi4-dec.h"
#include "psi4/libparallel/parallel.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libciomr/libciomr.h"
#include "psi4/libpsio/psio.hpp"
#include "psi4/libqt/qt.h"
#include "psi4/libmints/molecule.h"
#include "psi4/libmints/writer.h"
#include "psi4/libmints/writer_file_prefix.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/oeprop.h"
#include "psi4/libmints/local.h"
#include "psi4/libfunctional/superfunctional.h"
#include "psi4/libtrans/mospace.h"
#include "psi4/libtrans/integraltransform.h"
#include "psi4/libscf_solver/rhf.h"
#include "psi4/libdpd/dpd.h"
```

### Classes

- class `oepdev::WavefunctionUnion`  
*Union of two Wavefunction objects.*

### Namespaces

- `oepdev`  
*OEPEv module namespace.*



## Chapter 19

# Example Documentation

### 19.1 example\_cphf.cc

Shows how to use the `oepdev::CPHF` solver to compute molecular and LMO-distributed polarizabilities at RHF level of theory.

```
void example_cphf(std::shared_ptr<psi::Wavefunction> wfn, psi::Options& opt){  
    // build the solver  
    std::shared_ptr<oepdev::CPHF> solver = std::make_shared<oepdev::CPHF>(wfn, opt);  
  
    // run the solver to converge CPHF equations  
    solver->compute();  
  
    // print the LMO-distributed polarizabilities  
    for (int i=0; i<solver->nocc(); i++) {  
        solver->polarizability(i)->print();  
    }  
  
    // print the molecular polarizability  
    solver->polarizability()->print();  
  
    // grab 4th LMO-distributed polarizability and its associated LMO centroid  
    psi::SharedMatrix pol_4 = solver->polarizability(3);  
    psi::SharedVector rmo_4 = solver->lmo_centroid(3);  
};
```

### 19.2 example\_integrals\_iter.cc

Iterations over electron repulsion integrals in AO basis. This is an example of how to use

- the `oepdev::ShellCombinationsIterator` class
- the `oepdev::AOIntegralsIterator` class.

```
void iterate(std::shared_ptr<oepdev::IntegralFactory> ints)  
{  
    // Prepare for direct calculation of ERI's (shell by shell)  
    std::shared_ptr<psi::TwoBodyAOInt> tei(ints->eri());  
  
    // Grab the buffer where the integrals for a current shell will be placed  
    const double* buffer = tei->buffer();  
  
    // Create iterator to go through all shell quartet combinations  
    oepdev::SharedShellsIterator shellIter =  
        oepdev::ShellCombinationsIterator::build(ints, "ALL", 4);  
  
    // Iterate over shells, and then over all integrals in each shell quartet  
    for (shellIter->first(); shellIter->is_done() == false; shellIter->next())  
    {  
        // Compute all integrals between shells in the current quartet  
        shellIter->compute_shell(tei);  
    }  
}
```

```

// Create iterator to go through all integrals within a shell quartet
oepdev::SharedAOIntsIterator intsIter = shellIter->ao_iterator("ALL");

for (intsIter->first(); intsIter->is_done() == false; intsIter->next())
{
    // Grab current (ij|kl) indices here
    int i = intsIter->i();
    int j = intsIter->j();
    int k = intsIter->k();
    int l = intsIter->l();

    // Grab the (ij|kl) integral
    double integral = buffer[intsIter->index()];
}
}
}

```

### 19.3 example\_scf\_perturb.cc

Perturb HF Hamiltonian with external electrostatic potential. This is an example of how to use the [oepdev::RHF-Perturbed](#) class.

```

void scf_perturb(std::shared_ptr<psi::Wavefunction> wfn, psi::Options& opt)
{
    // Set up HF superfunctional
    std::shared_ptr<psi::SuperFunctional> func = oepdev::create_superfunctional
        ("HF", opt);

    // Initialize the perturbed wavefunction
    std::shared_ptr<oepdev::RHPerturbed> scf = std::make_shared<oepdev::RHPerturbed>(wfn, func, opt, wfn->
        psio());

    /* Perturb the system with the uniform electric field [Fx, Fy, Fz].
       Then, add two point charges of charge qi placed at [Rxi, Ryi, Rzi].
       Provide all these values in atomic units! */
    const double Fx = 0.04, Fy = 0.05, Fz = -0.09;
    const double Rx1= 0.00, Rx2= 1.30, Rx3= -1.00;
    const double Rx1= 0.10, Rx2=-0.30, Rx3= 3.50;
    const double q1 = 0.30, q2 =-0.09;

    scf->set_perturbation(Fx, Fy, Fz);          /* set it only once, setting it again will overwrite the
        field, not add */
    scf->set_perturbation(Rx1, Ry1, Rz1, q1);
    scf->set_perturbation(Rx2, Ry2, Rz2, q2); /* more charges can be added */

    // Solve perturbed SCF equations
    scf->compute_energy();

    // Grab some data
    double energy = scf->reference_energy();    // Total energy of the system
    std::shared_ptr<psi::Matrix> Da = scf->Da(); // One-particle density matrix

    /* Note that the external field and charges perturb only one-electron Hamiltonian.*/
}

```

# Index

- AllIAOIntegralsIterator\_2
  - oepdev::AllIAOIntegralsIterator\_2, [67](#)
- AllIAOIntegralsIterator\_4
  - oepdev::AllIAOIntegralsIterator\_4, [68](#), [69](#)
- AllIAOShellCombinationsIterator\_2
  - oepdev::AllIAOShellCombinationsIterator\_2, [70](#)
- AllIAOShellCombinationsIterator\_4
  - oepdev::AllIAOShellCombinationsIterator\_4, [72](#), [73](#)
- allocate
  - oepdev::GenEffPar, [113](#)
- ao\_iterator
  - oepdev::ShellCombinationsIterator, [162](#)
- average\_moment
  - The OEPDev Utilities, [54](#)
- build
  - oepdev::AOIntegralsIterator, [74](#)
  - oepdev::DMTPole, [89](#)
  - oepdev::Field3D, [107](#), [108](#)
  - oepdev::GenEffParFactory, [117](#)
  - oepdev::GeneralizedDensityFit, [120](#)
  - oepdev::OEPDevSolver, [132](#)
  - oepdev::OEPotential, [135](#)
  - oepdev::Points3DIterator, [140](#), [142](#)
  - oepdev::PointsCollection3D, [144](#)
  - oepdev::ShellCombinationsIterator, [162](#)
- CPHF
  - oepdev::CPHF, [82](#)
- Ca\_subset
  - oepdev::WavefunctionUnion, [183](#)
- Cb\_subset
  - oepdev::WavefunctionUnion, [183](#)
- compute
  - oepdev::DIISManager, [85](#)
  - oepdev::DoubleGeneralizedDensityFit, [92](#)
  - oepdev::GeneralizedDensityFit, [120](#)
  - oepdev::SingleGeneralizedDensityFit, [164](#)
  - oepdev::TwoBodyAOInt, [166](#), [167](#)
- compute\_benchmark
  - oepdev::ChargeTransferEnergySolver, [78](#)
  - oepdev::ElectrostaticEnergySolver, [96](#)
  - oepdev::OEPDevSolver, [132](#)
  - oepdev::RepulsionEnergySolver, [157](#)
- compute\_density\_matrix
  - oepdev::GenEffPar, [113](#), [114](#)
- compute\_oep\_based
  - oepdev::ChargeTransferEnergySolver, [79](#)
  - oepdev::ElectrostaticEnergySolver, [97](#)
  - oepdev::OEPDevSolver, [132](#)
  - oepdev::RepulsionEnergySolver, [157](#)
- compute\_shell
  - oepdev::AllIAOShellCombinationsIterator\_2, [71](#)
  - oepdev::AllIAOShellCombinationsIterator\_4, [73](#)
  - oepdev::ShellCombinationsIterator, [162](#)
  - oepdev::TwoElectronInt, [169](#)
- create\_superfunctional
  - The OEPDev Utilities, [55](#)
- d\_N\_n1\_n2
  - The Integral Package Library, [46](#)
- DIISManager
  - oepdev::DIISManager, [85](#)
- DMTPole
  - oepdev::DMTPole, [89](#)
- ESPSolver
  - oepdev::ESPSolver, [104](#)
- energy
  - oepdev::DMTPole, [89](#)
- extract\_monomer
  - The OEPDev Utilities, [55](#)
- Field3D
  - oepdev::Field3D, [107](#)
- include/oepdev\_files.h, [185](#)
- include/oepdev\_options.h, [185](#)
- index
  - oepdev::AllIAOIntegralsIterator\_2, [67](#)
  - oepdev::AllIAOIntegralsIterator\_4, [69](#)
- main.cc, [186](#)
- make\_mdh\_D1\_coeff
  - The Integral Package Library, [46](#)
- make\_mdh\_D2\_coeff
  - The Integral Package Library, [46](#)
- make\_mdh\_D2\_coeff\_explicit\_recursion
  - The Integral Package Library, [47](#)
- make\_mdh\_D3\_coeff
  - The Integral Package Library, [47](#)
- make\_mdh\_R\_coeff
  - The Integral Package Library, [48](#)
- make\_oeps3d
  - oepdev::OEPotential, [136](#)
- OEPDevSolver
  - oepdev::OEPDevSolver, [132](#)
- OEPotential
  - oepdev::OEPotential, [135](#)
- OEPotential3D

- The Three-Dimensional Vector Fields Library, 50
- oepdev, 59
  - psi, 63
- oepdev/lib3d/dmtp.h, 186
- oepdev/lib3d/esp.h, 187
- oepdev/libgefp/gefp.h, 187
- oepdev/libints/eri.h, 188
- oepdev/libints/recurr.h, 189
- oepdev/liboep/oep.h, 190
- oepdev/liboep/oep\_gdf.h, 191
- oepdev/libpsi/integral.h, 191
- oepdev/libpsi/potential.h, 192
- oepdev/libsolver/solver.h, 192
- oepdev/libtest/test.h, 193
- oepdev/libutil/diis.h, 193
- oepdev/libutil/integrals\_iter.h, 194
- oepdev/libutil/scf\_perturb.h, 194
- oepdev/libutil/unitary\_optimizer.h, 195
- oepdev/libutil/util.h, 196
- oepdev/libutil/wavefunction\_union.h, 197
- oepdev::ABCD, 65
- oepdev::AOIntegralsIterator, 73
  - build, 74
- oepdev::AbInitioPolarGEFactory, 65
- oepdev::AllAOIntegralsIterator\_2, 66
  - AllAOIntegralsIterator\_2, 67
  - index, 67
- oepdev::AllAOIntegralsIterator\_4, 68
  - AllAOIntegralsIterator\_4, 68, 69
  - index, 69
- oepdev::AllAOShellCombinationsIterator\_2, 69
  - AllAOShellCombinationsIterator\_2, 70
  - compute\_shell, 71
- oepdev::AllAOShellCombinationsIterator\_4, 71
  - AllAOShellCombinationsIterator\_4, 72, 73
  - compute\_shell, 73
- oepdev::CAMM, 75
- oepdev::CPHF, 79
  - CPHF, 82
- oepdev::ChargeTransferEnergyOEPotential, 75
- oepdev::ChargeTransferEnergySolver, 76
  - compute\_benchmark, 78
  - compute\_oep\_based, 79
- oepdev::CubePoints3DIterator, 83
- oepdev::CubePointsCollection3D, 84
- oepdev::DIISManager, 84
  - compute, 85
  - DIISManager, 85
  - put, 85
  - update, 85
- oepdev::DMTPole, 86
  - build, 89
  - DMTPole, 89
  - energy, 89
  - potential, 90
- oepdev::DoubleGeneralizedDensityFit, 90
  - compute, 92
- oepdev::EETCouplingOEPotential, 92
- oepdev::ERI\_1\_1, 98
- oepdev::ERI\_2\_2, 99
- oepdev::ERI\_3\_1, 101
- oepdev::ESPSolver, 102
  - ESPSolver, 104
- oepdev::ElectrostaticEnergyOEPotential, 93
- oepdev::ElectrostaticEnergySolver, 94
  - compute\_benchmark, 96
  - compute\_oep\_based, 97
- oepdev::ElectrostaticPotential3D, 97
- oepdev::FFAbInitioPolarGEFactory, 104
- oepdev::Field3D, 105
  - build, 107, 108
  - Field3D, 107
- oepdev::Fourier9, 108
- oepdev::GenEffFrag, 109
  - susceptibility, 110
- oepdev::GenEffPar, 111
  - allocate, 113
  - compute\_density\_matrix, 113, 114
  - set\_susceptibility, 114
  - susceptibility, 114, 115
- oepdev::GenEffParFactory, 116
  - build, 117
- oepdev::GeneralizedDensityFit, 118
  - build, 120
  - compute, 120
- oepdev::GeneralizedPolarGEFactory, 120
- oepdev::GeneralizedPolarGEFactory::StatisticalSet, 164
- oepdev::IntegralFactory, 125
- oepdev::LinearGradientNonUniformEFieldPolarGE-Factory, 126
- oepdev::LinearNonUniformEFieldPolarGEFactory, 127
- oepdev::LinearUniformEFieldPolarGEFactory, 128
- oepdev::MultipoleConvergence, 129
- oepdev::NonUniformEFieldPolarGEFactory, 130
- oepdev::OEPDevSolver, 130
  - build, 132
  - compute\_benchmark, 132
  - compute\_oep\_based, 132
  - OEPDevSolver, 132
- oepdev::OEType, 137
- oepdev::OEPotential, 133
  - build, 135
  - make\_oeps3d, 136
  - OEPotential, 135
- oepdev::OEPotential3D< T >, 136
- oepdev::PerturbCharges, 138
- oepdev::Points3DIterator, 139
  - build, 140, 142
  - Points3DIterator, 140
- oepdev::Points3DIterator::Point, 138
- oepdev::PointsCollection3D, 142
  - build, 144
  - PointsCollection3D, 143
- oepdev::PolarGEFactory, 144
- oepdev::PotentialInt, 146
  - PotentialInt, 146, 147



- set\_charge\_field, 147
- oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory, 147
- oepdev::QuadraticNonUniformEFieldPolarGEFactory, 149
- oepdev::QuadraticUniformEFieldPolarGEFactory, 149
- oepdev::RHFPerturbed, 158
- oepdev::RandomPoints3DIterator, 150
- oepdev::RandomPointsCollection3D, 152
- oepdev::RepulsionEnergyOEPotential, 152
- oepdev::RepulsionEnergySolver, 153
  - compute\_benchmark, 157
  - compute\_oep\_based, 157
- oepdev::ShellCombinationsIterator, 159
  - ao\_iterator, 162
  - build, 162
  - compute\_shell, 162
  - ShellCombinationsIterator, 161
- oepdev::SingleGeneralizedDensityFit, 163
  - compute, 164
- oepdev::TwoBodyAOInt, 166
  - compute, 166, 167
- oepdev::TwoElectronInt, 167
  - compute\_shell, 169
- oepdev::UniformEFieldPolarGEFactory, 169
- oepdev::UnitaryOptimizer, 170
  - UnitaryOptimizer, 173, 174
- oepdev::UnitaryOptimizer\_4\_2, 174
  - UnitaryOptimizer\_4\_2, 178
- oepdev::UnitaryTransformedMOPolarGEFactory, 178
- oepdev::WavefunctionUnion, 179
  - Ca\_subset, 183
  - Cb\_subset, 183
  - WavefunctionUnion, 183
- oepdev::test::Test, 165
- Points3DIterator
  - oepdev::Points3DIterator, 140
- PointsCollection3D
  - oepdev::PointsCollection3D, 143
- potential
  - oepdev::DMTPole, 90
- PotentialInt
  - oepdev::PotentialInt, 146, 147
- psi, 63
  - oepdev, 63
  - read\_options, 64
- put
  - oepdev::DIISManager, 85
- read\_options
  - psi, 64
- set\_charge\_field
  - oepdev::PotentialInt, 147
- set\_susceptibility
  - oepdev::GenEffPar, 114
- ShellCombinationsIterator
  - oepdev::ShellCombinationsIterator, 161
- solve\_scf
  - The OEPDev Utilities, 55
- susceptibility
  - oepdev::GenEffFrag, 110
  - oepdev::GenEffPar, 114, 115
- The Density Functional Theory Library, 52
- The Generalized Effective Fragment Potentials Library, 41
- The Generalized One-Electron Potentials Library, 39
- The Integral Package Library, 42
  - d\_N\_n1\_n2, 46
  - make\_mdh\_D1\_coeff, 46
  - make\_mdh\_D2\_coeff, 46
  - make\_mdh\_D2\_coeff\_explicit\_recursion, 47
  - make\_mdh\_D3\_coeff, 47
  - make\_mdh\_R\_coeff, 48
- The OEPDev Solver Library, 40
- The OEPDev Testing Platform Library, 57
- The OEPDev Utilities, 53
  - average\_moment, 54
  - create\_superfunctional, 55
  - extract\_monomer, 55
  - solve\_scf, 55
- The Three-Dimensional Vector Fields Library, 49
  - OEPotential3D, 50
- UnitaryOptimizer
  - oepdev::UnitaryOptimizer, 173, 174
- UnitaryOptimizer\_4\_2
  - oepdev::UnitaryOptimizer\_4\_2, 178
- update
  - oepdev::DIISManager, 85
- WavefunctionUnion
  - oepdev::WavefunctionUnion, 183