

OEP-Dev 1.0.2-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>GEFP-OEP</b>	<b>1</b>
<b>2</b>	<b>License</b>	<b>3</b>
<b>3</b>	<b>Introduction</b>	<b>5</b>
3.1	Research Project Methodology . . . . .	6
3.2	Expected Impact on the Development of Science, Civilization and Society . . . .	6
3.3	The OEPDev Code . . . . .	7
<b>4</b>	<b>OEP Design.</b>	<b>9</b>
4.1	OEP Classes . . . . .	9
4.1.1	Structure of possible OEP-based expressions and their unification . . . .	9
<b>5</b>	<b>List of One-Electron Potentials</b>	<b>11</b>
5.1	Electrostatic Energy OEP's . . . . .	11
5.2	Pauli Repulsion OEP's . . . . .	11
5.2.1	First-order contribution in overlap matrix expansion. . . . .	12
5.2.2	Second-order contribution in overlap matrix expansion. . . . .	12
5.3	Charge-Transfer Energy OEP's . . . . .	12
5.4	Excitonic Energy Transfer OEP's . . . . .	12
5.4.1	ET contributions. . . . .	12
5.4.2	HT contributions. . . . .	13
5.4.3	CT contributions. . . . .	13
5.5	Full HF Interaction OEP's . . . . .	13
<b>6</b>	<b>Density-fitting Specialized for OEP's</b>	<b>15</b>
6.1	Fitting in Complete Space . . . . .	15
6.2	Fitting in Incomplete Space . . . . .	16

<b>7</b>	<b>Implemented Models</b>	<b>19</b>
7.1	Target Properties . . . . .	19
7.2	Target, Benchmark and Competing Models . . . . .	19
<b>8</b>	<b>Contributing to OEP-Dev</b>	<b>21</b>
8.1	Main routine and libraries . . . . .	21
8.2	Header files in libraries . . . . .	21
8.3	Environmental variables . . . . .	22
8.4	Documenting the code . . . . .	22
8.5	Naming conventions . . . . .	23
8.6	Track timing when evaluating the code . . . . .	23
8.7	Clean memory between independent jobs . . . . .	24
8.8	Use Object-Oriented Programming . . . . .	24
<b>9</b>	<b>Advanced Usage</b>	<b>25</b>
9.1	Installation . . . . .	25
9.1.1	Preparing Psi4 . . . . .	25
9.1.2	Compiltation . . . . .	26
9.2	OEPDev Code Structure . . . . .	26
9.2.1	Main Routine . . . . .	27
9.2.2	Modules . . . . .	27
9.3	OEPDev Classes: Overview . . . . .	28
9.3.1	OEP Module . . . . .	28
9.3.1.1	OEPPotential . . . . .	28
9.3.1.2	GeneralizedDensityFit . . . . .	28
9.3.2	GEFP Module . . . . .	28
9.3.2.1	GenEffPar . . . . .	28
9.3.2.2	GenEffParFactory . . . . .	28
9.3.2.3	GenEffFrag . . . . .	28
9.3.3	OEPDev Solver Module . . . . .	28
9.3.3.1	OEPDevSolver . . . . .	28
9.4	Developing OEP's . . . . .	29
9.4.1	Drafting an OEP Subclass . . . . .	29
9.4.1.1	Implementing OEP Types . . . . .	30
9.4.1.2	Abstract Base . . . . .	31

<b>10 License</b>	<b>33</b>
<b>11 Module Index</b>	<b>35</b>
11.1 Modules . . . . .	35
<b>12 Namespace Index</b>	<b>37</b>
12.1 Namespace List . . . . .	37
<b>13 Hierarchical Index</b>	<b>39</b>
13.1 Class Hierarchy . . . . .	39
<b>14 Class Index</b>	<b>45</b>
14.1 Class List . . . . .	45
<b>15 File Index</b>	<b>51</b>
15.1 File List . . . . .	51
<b>16 Module Documentation</b>	<b>53</b>
16.1 The Generalized One-Electron Potentials Library . . . . .	53
16.1.1 Detailed Description . . . . .	53
16.2 The OEPDev Solver Library . . . . .	54
16.2.1 Detailed Description . . . . .	54
16.3 The Generalized Effective Fragment Potentials Library . . . . .	55
16.3.1 Detailed Description . . . . .	56
16.4 The Integral Package Library . . . . .	57
16.4.1 Detailed Description . . . . .	58
16.4.2 Hermite Operators . . . . .	59
16.4.2.1 Polynomial Expansions as Hermite Series . . . . .	59
16.4.3 One-Body Integrals over Hermite Functions . . . . .	60
16.4.4 Two-Body Integrals over Hermite Functions . . . . .	61
16.4.5 The R(N,L,M) Coefficients . . . . .	61
16.4.6 Function Documentation . . . . .	62
16.4.6.1 d_N_n1_n2() . . . . .	62
16.4.6.2 make_mdh_D1_coeff() . . . . .	62
16.4.6.3 make_mdh_D2_coeff() . . . . .	63
16.4.6.4 make_mdh_D2_coeff_explicit_recursion() . . . . .	63
16.4.6.5 make_mdh_D3_coeff() . . . . .	64

16.4.6.6	make_mdh_R_coeff()	65
16.5	The Three-Dimensional Vector Fields Library	67
16.5.1	Detailed Description	68
16.5.2	Function Documentation	68
16.5.2.1	OEPotential3D() [1/2]	68
16.5.2.2	OEPotential3D() [2/2]	69
16.6	The Density Functional Theory Library	70
16.7	The OEPDev Utilities	71
16.7.1	Theory	74
16.7.2	Detailed Description	76
16.7.3	Function Documentation	76
16.7.3.1	_calculate_DFI_Vel()	76
16.7.3.2	ao_rotation_matrix()	77
16.7.3.3	average_moment()	77
16.7.3.4	calculate_der_D()	78
16.7.3.5	calculate_DFI_Vel_J()	78
16.7.3.6	calculate_DFI_Vel_JK()	78
16.7.3.7	calculate_e_xc()	79
16.7.3.8	calculate_JK()	79
16.7.3.9	calculate_JK_r()	80
16.7.3.10	compute_distance()	80
16.7.3.11	create_superfunctional()	81
16.7.3.12	extract_monomer()	81
16.7.3.13	matrix_power_derivative()	82
16.7.3.14	populate()	82
16.7.3.15	r6()	83
16.7.3.16	solve_scf()	83
16.7.3.17	solve_scf_sad()	84
16.8	The OEPDev Testing Platform Library	86
16.8.1	Detailed Description	86
<b>17</b>	<b>Namespace Documentation</b>	<b>87</b>
17.1	gefp.basis.optimize Namespace Reference	87
17.1.1	Detailed Description	87
17.2	gefp.density.dfi Namespace Reference	87

17.2.1 Detailed Description . . . . .	88
17.3 <a href="#">gefp.density.population Namespace Reference</a> . . . . .	88
17.3.1 Detailed Description . . . . .	89
17.3.2 Function Documentation . . . . .	89
17.3.2.1 <a href="#">atomic_charges()</a> . . . . .	89
17.4 <a href="#">oepdev Namespace Reference</a> . . . . .	89
17.4.1 Theory . . . . .	96
17.4.2 Detailed Description . . . . .	98
17.5 <a href="#">psi Namespace Reference</a> . . . . .	98
17.5.1 Detailed Description . . . . .	98
17.5.2 Function Documentation . . . . .	98
17.5.2.1 <a href="#">oepdev()</a> . . . . .	99
17.5.2.2 <a href="#">read_options()</a> . . . . .	99
<b>18 Class Documentation</b> . . . . .	<b>101</b>
18.1 <a href="#">gefp.density.dms._DMS_SCF_Procedure Class Reference</a> . . . . .	101
18.2 <a href="#">gefp.density.dms._Global_Settings_DMSFit Class Reference</a> . . . . .	101
18.2.1 Detailed Description . . . . .	102
18.3 <a href="#">gefp.density.functional.A_V1_MEDI_XCFunctional Class Reference</a> . . . . .	102
18.3.1 Detailed Description . . . . .	103
18.4 <a href="#">gefp.density.functional.A_V2_MEDI_XCFunctional Class Reference</a> . . . . .	103
18.4.1 Detailed Description . . . . .	104
18.5 <a href="#">oepdev::ABCD Struct Reference</a> . . . . .	104
18.5.1 Detailed Description . . . . .	104
18.6 <a href="#">oepdev::AbInitioPolarGEFactory Class Reference</a> . . . . .	105
18.6.1 Detailed Description . . . . .	105
18.7 <a href="#">gefp.density.dms.Aggregate Class Reference</a> . . . . .	106
18.8 <a href="#">oepdev::AllAOIntegralsIterator_2 Class Reference</a> . . . . .	106
18.8.1 Detailed Description . . . . .	107
18.8.2 Constructor & Destructor Documentation . . . . .	107
18.8.2.1 <a href="#">AllAOIntegralsIterator_2()</a> [1/2] . . . . .	107
18.8.2.2 <a href="#">AllAOIntegralsIterator_2()</a> [2/2] . . . . .	108
18.8.3 Member Function Documentation . . . . .	108
18.8.3.1 <a href="#">index()</a> . . . . .	108
18.9 <a href="#">oepdev::AllAOIntegralsIterator_4 Class Reference</a> . . . . .	108

18.9.1 Detailed Description . . . . .	109
18.9.2 Constructor & Destructor Documentation . . . . .	109
18.9.2.1 AllAOIntegralsIterator_4() [1/2] . . . . .	109
18.9.2.2 AllAOIntegralsIterator_4() [2/2] . . . . .	110
18.9.3 Member Function Documentation . . . . .	110
18.9.3.1 index() . . . . .	110
18.10 oepdev::AllAOShellCombinationsIterator_2 Class Reference . . . . .	110
18.10.1 Detailed Description . . . . .	111
18.10.2 Constructor & Destructor Documentation . . . . .	111
18.10.2.1 AllAOShellCombinationsIterator_2() [1/5] . . . . .	111
18.10.2.2 AllAOShellCombinationsIterator_2() [2/5] . . . . .	112
18.10.2.3 AllAOShellCombinationsIterator_2() [3/5] . . . . .	112
18.10.2.4 AllAOShellCombinationsIterator_2() [4/5] . . . . .	112
18.10.2.5 AllAOShellCombinationsIterator_2() [5/5] . . . . .	112
18.10.3 Member Function Documentation . . . . .	112
18.10.3.1 compute_shell() . . . . .	112
18.11 oepdev::AllAOShellCombinationsIterator_4 Class Reference . . . . .	113
18.11.1 Detailed Description . . . . .	114
18.11.2 Constructor & Destructor Documentation . . . . .	114
18.11.2.1 AllAOShellCombinationsIterator_4() [1/5] . . . . .	114
18.11.2.2 AllAOShellCombinationsIterator_4() [2/5] . . . . .	114
18.11.2.3 AllAOShellCombinationsIterator_4() [3/5] . . . . .	115
18.11.2.4 AllAOShellCombinationsIterator_4() [4/5] . . . . .	115
18.11.2.5 AllAOShellCombinationsIterator_4() [5/5] . . . . .	115
18.11.3 Member Function Documentation . . . . .	115
18.11.3.1 compute_shell() [1/2] . . . . .	115
18.11.3.2 compute_shell() [2/2] . . . . .	116
18.12 oepdev::AOIntegralsIterator Class Reference . . . . .	116
18.12.1 Detailed Description . . . . .	117
18.12.2 Member Function Documentation . . . . .	117
18.12.2.1 build() [1/2] . . . . .	117
18.12.2.2 build() [2/2] . . . . .	118
18.13 gefp.density.dms.Basic_DMS Class Reference . . . . .	118
18.13.1 Detailed Description . . . . .	118
18.14 gefp.density.dms.BasicD_DMS Class Reference . . . . .	119



18.14.1 Detailed Description . . . . .	119
18.15gefp.density.functional.BBC1_XCFunctional Class Reference . . . . .	119
18.15.1 Detailed Description . . . . .	120
18.16gefp.density.functional.BBC2_XCFunctional Class Reference . . . . .	120
18.16.1 Detailed Description . . . . .	121
18.17oepdev::CAMM Class Reference . . . . .	121
18.17.1 Detailed Description . . . . .	121
18.18oepdev::ChargeTransferEnergyOEPotential Class Reference . . . . .	122
18.18.1 Detailed Description . . . . .	123
18.19oepdev::ChargeTransferEnergySolver Class Reference . . . . .	123
18.19.1 Detailed Description . . . . .	124
18.19.2 Member Function Documentation . . . . .	126
18.19.2.1 compute_benchmark() . . . . .	126
18.19.2.2 compute_oep_based() . . . . .	126
18.20gefp.density.ci.CIS_CIWavefunction Class Reference . . . . .	127
18.21oepdev::CISComputer Class Reference . . . . .	127
18.21.1 Detailed Description . . . . .	130
18.21.2 Member Function Documentation . . . . .	130
18.21.2.1 build() . . . . .	131
18.21.3 Member Data Documentation . . . . .	134
18.21.3.1 nmo_ . . . . .	134
18.22oepdev::CISData Struct Reference . . . . .	134
18.23gefp.density.ci.CIWavefunction Class Reference . . . . .	135
18.24gefp.density.dms.Computer Class Reference . . . . .	136
18.24.1 Detailed Description . . . . .	136
18.25oepdev::CPHF Class Reference . . . . .	137
18.25.1 Detailed Description . . . . .	140
18.25.2 Constructor & Destructor Documentation . . . . .	140
18.25.2.1 CPHF() . . . . .	140
18.26oepdev::CubePoints3DIterator Class Reference . . . . .	140
18.26.1 Detailed Description . . . . .	141
18.27oepdev::CubePointsCollection3D Class Reference . . . . .	142
18.27.1 Detailed Description . . . . .	142
18.28oepdev::DavidsonLiu Class Reference . . . . .	142
18.28.1 Detailed Description . . . . .	144

18.29	<a href="#">gefp.density.opdm.Density Class Reference</a>	146
18.29.1	<a href="#">Detailed Description</a>	147
18.29.2	<a href="#">Member Function Documentation</a>	148
18.29.2.1	<a href="#">natural_orbitals()</a>	148
18.30	<a href="#">gefp.density.partitioning.DensityDecomposition Class Reference</a>	150
18.30.1	<a href="#">Detailed Description</a>	151
18.30.2	<a href="#">Member Function Documentation</a>	153
18.30.2.1	<a href="#">compute()</a>	153
18.30.2.2	<a href="#">deformation_density()</a>	153
18.31	<a href="#">gefp.density.opdm.DensityProjection Class Reference</a>	154
18.31.1	<a href="#">Detailed Description</a>	154
18.32	<a href="#">gefp.basis.optimize.DFBasis Class Reference</a>	154
18.32.1	<a href="#">Detailed Description</a>	155
18.33	<a href="#">gefp.basis.optimize.DFBasisOptimizer Class Reference</a>	155
18.33.1	<a href="#">Detailed Description</a>	156
18.34	<a href="#">gefp.density.dfi.DFI Class Reference</a>	156
18.34.1	<a href="#">Detailed Description</a>	157
18.34.2	<a href="#">Member Function Documentation</a>	157
18.34.2.1	<a href="#">run()</a>	157
18.35	<a href="#">gefp.density.dfi.DFI_J Class Reference</a>	158
18.36	<a href="#">gefp.density.dfi.DFI_JK Class Reference</a>	158
18.37	<a href="#">oepdev::DIISManager Class Reference</a>	159
18.37.1	<a href="#">Detailed Description</a>	159
18.37.2	<a href="#">Constructor &amp; Destructor Documentation</a>	159
18.37.2.1	<a href="#">DIISManager()</a>	159
18.37.3	<a href="#">Member Function Documentation</a>	160
18.37.3.1	<a href="#">compute()</a>	160
18.37.3.2	<a href="#">put()</a>	160
18.37.3.3	<a href="#">update()</a>	160
18.38	<a href="#">gefp.density.dmft.DMFT Class Reference</a>	160
18.38.1	<a href="#">Detailed Description</a>	162
18.38.2	<a href="#">Member Function Documentation</a>	163
18.38.2.1	<a href="#">create()</a>	163
18.38.2.2	<a href="#">run()</a>	163
18.39	<a href="#">gefp.density.dmft.DMFT_AO Class Reference</a>	164

18.40gefp.density.dmft.DMFT_MO Class Reference . . . . .	164
18.41gefp.density.dmft.DMFT_NC Class Reference . . . . .	165
18.42gefp.density.dmft.DMFT_PC Class Reference . . . . .	165
18.43gefp.density.dmft.DMFT_ProjD Class Reference . . . . .	166
18.44gefp.density.dmft.DMFT_ProjP Class Reference . . . . .	167
18.45gefp.density.dms.DMS Class Reference . . . . .	167
18.45.1 Detailed Description . . . . .	168
18.46gefp.density.dms.DMSFit Class Reference . . . . .	169
18.46.1 Detailed Description . . . . .	170
18.47oepdev::DMTPole Class Reference . . . . .	170
18.47.1 Detailed Description . . . . .	175
18.47.2 Constructor & Destructor Documentation . . . . .	175
18.47.2.1 DMTPole() [1/2] . . . . .	175
18.47.2.2 DMTPole() [2/2] . . . . .	175
18.47.3 Member Function Documentation . . . . .	176
18.47.3.1 build() . . . . .	176
18.47.3.2 compute() [1/2] . . . . .	176
18.47.3.3 compute() [2/2] . . . . .	176
18.47.3.4 determine_dmtp_convergence_level() . . . . .	177
18.47.3.5 empty() . . . . .	177
18.47.3.6 energy() . . . . .	177
18.47.3.7 field() . . . . .	178
18.47.3.8 potential() . . . . .	178
18.47.3.9 recenter() . . . . .	179
18.47.3.10 rotate() . . . . .	180
18.47.3.11 superimpose() . . . . .	181
18.47.4 Friends And Related Function Documentation . . . . .	181
18.47.4.1 MultipoleConvergence . . . . .	182
18.48oepdev::DoubleGeneralizedDensityFit Class Reference . . . . .	182
18.48.1 Detailed Description . . . . .	182
18.48.2 Determination of the OEP matrix . . . . .	182
18.48.2.1 Theory behind the double GDF scheme . . . . .	183
18.48.3 Member Function Documentation . . . . .	184
18.48.3.1 compute() . . . . .	184
18.49gefp.density.opdm.Dset_DensityProjection Class Reference . . . . .	184

18.49.1 Detailed Description . . . . .	185
18.50oepdev::EETCouplingOEPotential Class Reference . . . . .	185
18.50.1 Detailed Description . . . . .	186
18.51oepdev::EETCouplingSolver Class Reference . . . . .	186
18.51.1 Detailed Description . . . . .	187
18.51.2 Member Function Documentation . . . . .	192
18.51.2.1 compute_benchmark() . . . . .	192
18.51.2.2 compute_oep_based() . . . . .	192
18.52oepdev::EFP2_GEFactory Class Reference . . . . .	193
18.52.1 Detailed Description . . . . .	194
18.53gefp.density.dms.EFP_DMSFit Class Reference . . . . .	194
18.53.1 Detailed Description . . . . .	195
18.54gefp.density.dmt.ElectronCorrelation Class Reference . . . . .	195
18.54.1 Detailed Description . . . . .	195
18.55oepdev::ElectrostaticEnergyOEPotential Class Reference . . . . .	195
18.55.1 Detailed Description . . . . .	196
18.56oepdev::ElectrostaticEnergySolver Class Reference . . . . .	196
18.56.1 Detailed Description . . . . .	197
18.56.2 Member Function Documentation . . . . .	199
18.56.2.1 compute_benchmark() . . . . .	199
18.56.2.2 compute_oep_based() . . . . .	199
18.57oepdev::ElectrostaticPotential3D Class Reference . . . . .	200
18.57.1 Detailed Description . . . . .	201
18.58gefp.core.driver.Entry Class Reference . . . . .	201
18.59oepdev::ERI_1_1 Class Reference . . . . .	202
18.59.1 Detailed Description . . . . .	202
18.59.2 Implementation . . . . .	203
18.60oepdev::ERI_2_2 Class Reference . . . . .	203
18.60.1 Detailed Description . . . . .	204
18.60.2 Implementation . . . . .	204
18.61oepdev::ERI_3_1 Class Reference . . . . .	205
18.61.1 Detailed Description . . . . .	206
18.61.2 Implementation . . . . .	206
18.62oepdev::ESPSolver Class Reference . . . . .	206
18.62.1 Detailed Description . . . . .	207

18.62.2 Constructor & Destructor Documentation . . . . .	208
18.62.2.1 ESPSolver() [1/2] . . . . .	208
18.62.2.2 ESPSolver() [2/2] . . . . .	209
18.63gefp.density.dms.ExternalField_EFP_DMSFit Class Reference . . . . .	209
18.64oepdev::FFAbInitioPolarGEFactory Class Reference . . . . .	210
18.64.1 Detailed Description . . . . .	210
18.65oepdev::Field3D Class Reference . . . . .	211
18.65.1 Detailed Description . . . . .	213
18.65.2 Constructor & Destructor Documentation . . . . .	213
18.65.2.1 Field3D() . . . . .	214
18.65.3 Member Function Documentation . . . . .	214
18.65.3.1 build() [1/2] . . . . .	214
18.65.3.2 build() [2/2] . . . . .	214
18.66gefp.density.dms.Field_DMS Class Reference . . . . .	215
18.66.1 Detailed Description . . . . .	216
18.67oepdev::Fourier9 Struct Reference . . . . .	216
18.67.1 Detailed Description . . . . .	216
18.68oepdev::GenEffFrag Class Reference . . . . .	216
18.68.1 Detailed Description . . . . .	219
18.68.2 Member Function Documentation . . . . .	220
18.68.2.1 compute_energy() . . . . .	220
18.68.2.2 compute_many_body_energy() . . . . .	220
18.68.2.3 energy() . . . . .	220
18.68.2.4 susceptibility() [1/3] . . . . .	221
18.68.2.5 susceptibility() [2/3] . . . . .	221
18.68.2.6 susceptibility() [3/3] . . . . .	222
18.69oepdev::GenEffPar Class Reference . . . . .	222
18.69.1 Detailed Description . . . . .	226
18.69.2 Member Function Documentation . . . . .	226
18.69.2.1 allocate() . . . . .	226
18.69.2.2 basisset() . . . . .	227
18.69.2.3 compute_density_matrix() [1/4] . . . . .	227
18.69.2.4 compute_density_matrix() [2/4] . . . . .	227
18.69.2.5 compute_density_matrix() [3/4] . . . . .	228
18.69.2.6 compute_density_matrix() [4/4] . . . . .	228

18.69.2.7 dmtip()	228
18.69.2.8 dpol()	229
18.69.2.9 matrix()	229
18.69.2.10 rotate()	229
18.69.2.11 set_basiset()	230
18.69.2.12 set_dmtip()	230
18.69.2.13 set_dpol()	230
18.69.2.14 set_matrix()	230
18.69.2.15 set_susceptibility()	231
18.69.2.16 set_vector()	231
18.69.2.17 superimpose()	232
18.69.2.18 susceptibility() [1/3]	232
18.69.2.19 susceptibility() [2/3]	232
18.69.2.20 susceptibility() [3/3]	233
18.69.2.21 translate()	233
18.69.2.22 vector()	234
18.70 oepdev::GenEffParFactory Class Reference	234
18.70.1 Detailed Description	236
18.70.2 Member Function Documentation	236
18.70.2.1 build()	237
18.71 oepdev::GeneralizedDensityFit Class Reference	238
18.71.1 Detailed Description	239
18.71.2 Member Function Documentation	239
18.71.2.1 build() [1/2]	240
18.71.2.2 build() [2/2]	240
18.71.2.3 compute()	240
18.72 oepdev::GeneralizedPolarGEFactory Class Reference	241
18.72.1 Detailed Description	244
18.73 gefp.math.orthonorm.GrammSchmidt Class Reference	246
18.74 oepdev::GramSchmidt Class Reference	247
18.74.1 Detailed Description	248
18.74.2 Constructor & Destructor Documentation	248
18.74.2.1 GramSchmidt() [1/2]	248
18.74.2.2 GramSchmidt() [2/2]	248
18.74.3 Member Function Documentation	249

18.74.3.1 projection()	249
18.75gefp.density.functional.GU_XCFunctional Class Reference	249
18.75.1 Detailed Description	250
18.76gefp.density.parameters.Guess Class Reference	250
18.76.1 Detailed Description	251
18.77gefp.density.ci.HF_CIWavefunction Class Reference	251
18.78gefp.density.functional.HF_XCFunctional Class Reference	251
18.78.1 Detailed Description	252
18.79gefp.density.functional.IDF1_XCFunctional Class Reference	252
18.79.1 Detailed Description	253
18.80oepdev::IntegralFactory Class Reference	253
18.80.1 Detailed Description	254
18.81gefp.density.functional.Interpolating_XCFunctional Class Reference	254
18.81.1 Detailed Description	255
18.82gefp.density.functional.JKOnly_Interpolating_XCFunctional Class Reference	255
18.82.1 Detailed Description	256
18.83oepdev::KabschSuperimposer Class Reference	256
18.83.1 Detailed Description	257
18.83.2 Member Function Documentation	258
18.83.2.1 compute() [1/2]	258
18.83.2.2 compute() [2/2]	258
18.84oepdev::LinearGradientNonUniformEFieldPolarGEFactory Class Reference	258
18.84.1 Detailed Description	259
18.85oepdev::LinearNonUniformEFieldPolarGEFactory Class Reference	260
18.85.1 Detailed Description	260
18.86oepdev::LinearUniformEFieldPolarGEFactory Class Reference	261
18.86.1 Detailed Description	261
18.87gefp.density.population.Loc Class Reference	262
18.88gefp.density.parameters.Matrix_Guess Class Reference	262
18.89gefp.density.functional.MBB_XCFunctional Class Reference	263
18.89.1 Detailed Description	263
18.90gefp.density.functional.MEDI_XCFunctional Class Reference	263
18.90.1 Detailed Description	264
18.91oepdev::MultipoleConvergence Class Reference	264
18.91.1 Detailed Description	266

18.91.2 Member Enumeration Documentation . . . . .	266
18.91.2.1 ConvergenceLevel . . . . .	266
18.91.2.2 Property . . . . .	266
18.91.3 Constructor & Destructor Documentation . . . . .	267
18.91.3.1 MultipoleConvergence() . . . . .	267
18.91.4 Member Function Documentation . . . . .	267
18.91.4.1 compute() [1/2] . . . . .	267
18.91.4.2 compute() [2/2] . . . . .	267
18.91.4.3 level() . . . . .	268
18.92gefp.density.parameters.NC_Guess Class Reference . . . . .	268
18.93gefp.density.dms.New2_DMS Class Reference . . . . .	269
18.93.1 Detailed Description . . . . .	269
18.94gefp.density.dms.New_DMS Class Reference . . . . .	270
18.94.1 Detailed Description . . . . .	270
18.95oepdev::NonUniformEFieldPolarGEFactory Class Reference . . . . .	270
18.95.1 Detailed Description . . . . .	271
18.96gefp.basis.optimize.OEP Class Reference . . . . .	271
18.96.1 Detailed Description . . . . .	272
18.97gefp.basis.optimize.OEP_CT Class Reference . . . . .	272
18.97.1 Detailed Description . . . . .	272
18.98oepdev::OEP_EFP2_GEFactory Class Reference . . . . .	273
18.98.1 Detailed Description . . . . .	273
18.99gefp.basis.optimize.OEP_FockLike Class Reference . . . . .	273
18.99.1 Detailed Description . . . . .	274
18.100gefp.basis.optimize.OEP_Pauli Class Reference . . . . .	274
18.100.1 Detailed Description . . . . .	275
18.101oepdev::OEPDevSolver Class Reference . . . . .	275
18.101.1 Detailed Description . . . . .	276
18.101.2 Constructor & Destructor Documentation . . . . .	283
18.101.2.1 OEPDevSolver() . . . . .	283
18.101.3 Member Function Documentation . . . . .	283
18.101.3.1 build() . . . . .	283
18.101.3.2 compute_benchmark() . . . . .	284
18.101.3.3 compute_oep_based() . . . . .	284
18.102oepdev::OEPotential Class Reference . . . . .	284



18.102.Detailed Description . . . . .	287
18.102.Constructor & Destructor Documentation . . . . .	287
18.102.2.OEPotential() [1/2] . . . . .	287
18.102.2.OEPotential() [2/2] . . . . .	288
18.102.3.Member Function Documentation . . . . .	288
18.102.3.1.build() [1/2] . . . . .	288
18.102.3.2.build() [2/2] . . . . .	288
18.102.3.3.make_oeps3d() . . . . .	289
18.100epdev::OEPotential3D< T > Class Template Reference . . . . .	289
18.103.Detailed Description . . . . .	290
18.104efp.density.dmft.OEProp Class Reference . . . . .	291
18.105epdev::OEPTYPE Struct Reference . . . . .	291
18.106efp.density.dms.OtherComputer Class Reference . . . . .	292
18.107efp.density.functional.P_V2.MEDI_XCFunctional Class Reference . . . . .	292
18.107.Detailed Description . . . . .	293
18.108efp.core.driver.PadeApproximant_2D Class Reference . . . . .	293
18.108.Detailed Description . . . . .	294
18.109epdev::PerturbCharges Struct Reference . . . . .	294
18.109.Detailed Description . . . . .	294
18.110epdev::Points3DIterator::Point Struct Reference . . . . .	295
18.110epdev::Points3DIterator Class Reference . . . . .	295
18.111.Detailed Description . . . . .	296
18.111.Constructor & Destructor Documentation . . . . .	297
18.111.2.1.Points3DIterator() . . . . .	297
18.111.3.Member Function Documentation . . . . .	297
18.111.3.1.build() [1/3] . . . . .	297
18.111.3.2.build() [2/3] . . . . .	298
18.111.3.3.build() [3/3] . . . . .	298
18.110epdev::PointsCollection3D Class Reference . . . . .	298
18.112.Detailed Description . . . . .	300
18.112.Constructor & Destructor Documentation . . . . .	300
18.112.2.1.PointsCollection3D() . . . . .	300
18.112.3.Member Function Documentation . . . . .	300
18.112.3.1.build() [1/3] . . . . .	300
18.112.3.2.build() [2/3] . . . . .	301

18.112.3.3	build() [3/3]	301
18.110	epdev::PolarGEFactory Class Reference	302
18.113	Detailed Description	303
18.114	epdev::PotentialInt Class Reference	303
18.114	Constructor & Destructor Documentation	304
18.114.1.1	PotentialInt() [1/3]	304
18.114.1.2	PotentialInt() [2/3]	304
18.114.1.3	PotentialInt() [3/3]	305
18.114.2	Member Function Documentation	305
18.114.2.1	set_charge_field()	305
18.115	efp.density.dms.Property_Computer Class Reference	306
18.116	efp.density.opdm.Pset_DensityProjection Class Reference	307
18.116	Detailed Description	307
18.117	epdev::QuadraticGradientNonUniformEFieldPolarGEFactory Class Reference	307
18.117	Detailed Description	308
18.118	epdev::QuadraticNonUniformEFieldPolarGEFactory Class Reference	309
18.118	Detailed Description	309
18.119	epdev::QuadraticUniformEFieldPolarGEFactory Class Reference	310
18.119	Detailed Description	310
18.120	epdev::R_CISComputer Class Reference	311
18.120	epdev::R_CISComputer_Direct Class Reference	311
18.120	epdev::R_CISComputer_DL Class Reference	312
18.122	Detailed Description	313
18.123	epdev::R_CISComputer_Explicit Class Reference	314
18.124	epdev::RandomPoints3DIterator Class Reference	315
18.124	Detailed Description	316
18.125	epdev::RandomPointsCollection3D Class Reference	316
18.125	Detailed Description	317
18.126	efp.density.ci.Reference_SlaterDeterminant Class Reference	317
18.127	epdev::RepulsionEnergyOEPotential Class Reference	317
18.127	Detailed Description	318
18.128	epdev::RepulsionEnergySolver Class Reference	318
18.128	Detailed Description	319
18.128.2	Member Function Documentation	324
18.128.2.1	compute_benchmark()	324

18.128.2.2.compute_oep_based()	324
18.129.epdev::RHPerturbed Class Reference	325
18.129.Detailed Description	326
18.130.gefp.density.dms.Rotation_DMSFit Class Reference	327
18.130.Detailed Description	327
18.131.gefp.density.rvs.RVS Class Reference	327
18.131.Detailed Description	328
18.132.gefp.density.dfi.SCF Class Reference	329
18.132.Detailed Description	330
18.133.epdev::ShellCombinationsIterator Class Reference	330
18.133.Detailed Description	332
18.133.Constructor & Destructor Documentation	332
18.133.2.1.ShellCombinationsIterator()	332
18.133.3.Member Function Documentation	332
18.133.3.1ao_iterator()	332
18.133.3.2build() [1/2]	333
18.133.3.3build() [2/2]	333
18.133.3.4compute_shell() [1/2]	334
18.133.3.5compute_shell() [2/2]	334
18.134.gefp.density.dms.SimpleComputer Class Reference	335
18.135.gefp.density.ci.Single_SlaterDeterminant Class Reference	335
18.136.epdev::SingleGeneralizedDensityFit Class Reference	336
18.136.Detailed Description	336
18.136.Determination of the OEP matrix	336
18.136.3.Member Function Documentation	337
18.136.3.1.compute()	337
18.137.gefp.density.ci.SlaterDeterminant Class Reference	337
18.138.epdev::GeneralizedPolarGEFactory::StatisticalSet Struct Reference	338
18.139.gefp.math.matrix.Superimposer Class Reference	339
18.139.Detailed Description	339
18.139.Constructor & Destructor Documentation	339
18.139.2.1_init_()	340
18.139.3.Member Function Documentation	340
18.139.3.1.set()	340
18.140.epdev::test::Test Class Reference	340

18.140	epdev::TIData Class Reference . . . . .	343
18.141	Detailed Description . . . . .	345
18.141.1	Member Function Documentation . . . . .	347
18.141.2.1	coupling_direct() . . . . .	347
18.141.2.2	coupling_direct_coul() . . . . .	347
18.141.2.3	coupling_direct_exch() . . . . .	348
18.141.2.4	coupling_indirect() . . . . .	348
18.141.2.5	coupling_indirect_ti2() . . . . .	348
18.141.2.6	coupling_indirect_ti3() . . . . .	348
18.141.2.7	coupling_total() . . . . .	349
18.141.2.8	coupling_trcamm() . . . . .	349
18.141.2.9	overlap_corrected() . . . . .	349
18.141.2.10	overlap_corrected_direct() [1/2] . . . . .	350
18.141.2.11	overlap_corrected_direct() [2/2] . . . . .	351
18.141.2.12	overlap_corrected_indirect() . . . . .	351
18.141.3	Member Data Documentation . . . . .	351
18.141.3.1	v0 . . . . .	351
18.142	efp.density.dms.Translation.DMSFit Class Reference . . . . .	352
18.142	Detailed Description . . . . .	353
18.143	epdev::TwoBodyAOInt Class Reference . . . . .	353
18.143	Member Function Documentation . . . . .	354
18.143.1.1	compute() [1/2] . . . . .	354
18.143.1.2	compute() [2/2] . . . . .	354
18.144	epdev::TwoElectronInt Class Reference . . . . .	354
18.144	Detailed Description . . . . .	356
18.144.1	Member Function Documentation . . . . .	356
18.144.2.1	compute_shell() . . . . .	357
18.145	epdev::U_CISComputer Class Reference . . . . .	357
18.146	epdev::U_CISComputer_DL Class Reference . . . . .	358
18.146	Detailed Description . . . . .	358
18.147	epdev::U_CISComputer_Explicit Class Reference . . . . .	359
18.148	epdev::UniformEFieldPolarGEFactory Class Reference . . . . .	360
18.148	Detailed Description . . . . .	361
18.149	epdev::UnitaryOptimizer Class Reference . . . . .	361
18.149	Detailed Description . . . . .	364

18.149.2	Constructor & Destructor Documentation	366
18.149.2.1	UnitaryOptimizer() [1/3]	366
18.149.2.2	UnitaryOptimizer() [2/3]	366
18.149.2.3	UnitaryOptimizer() [3/3]	367
18.150	gefp.core.utilities.UnitaryOptimizer Class Reference	367
18.150.1	Detailed Description	368
18.150.2	Constructor & Destructor Documentation	368
18.150.2.1	__init__()	368
18.150.3	Member Function Documentation	368
18.150.3.1	maximize()	368
18.150.3.2	minimize()	369
18.150.3.3	un()	369
18.150.3.4	__Z__()	369
18.151	gefpdev::UnitaryOptimizer_4_2 Class Reference	369
18.151.1	Detailed Description	372
18.151.2	Constructor & Destructor Documentation	374
18.151.2.1	UnitaryOptimizer_4_2() [1/2]	374
18.151.2.2	UnitaryOptimizer_4_2() [2/2]	374
18.152	gefp.core.utilities.UnitaryOptimizer_4_2 Class Reference	375
18.152.1	Detailed Description	375
18.152.2	Constructor & Destructor Documentation	376
18.152.2.1	__init__()	376
18.152.3	Member Function Documentation	376
18.152.3.1	maximize()	376
18.152.3.2	minimize()	376
18.152.3.3	un()	376
18.152.3.4	__Z__()	377
18.153	gefpdev::UnitaryTransformedMOPolarGEFactory Class Reference	377
18.153.1	Detailed Description	377
18.154	gefp.core.driver.UniversalSurface Class Reference	378
18.154.1	Member Data Documentation	378
18.154.1.1	par_descr_fci_sto3g_1	378
18.154.1.2	par_descr_fci_sto3g_2	379
18.154.1.3	par_full_d_fci_sto3g_1	379
18.155	gefp.density.functional.V1_MEDI_XCFunctional Class Reference	379

18.155. Detailed Description . . . . .	380
18.156. <a href="#">gefp.density.functional.V2.MEDI_XCFunctional Class Reference</a> . . . . .	380
18.156. Detailed Description . . . . .	381
18.157. <a href="#">oepdev::WavefunctionUnion Class Reference</a> . . . . .	382
18.157. Detailed Description . . . . .	385
18.157. <a href="#">Constructor &amp; Destructor Documentation</a> . . . . .	387
18.157.2. <a href="#">WavefunctionUnion()</a> [1/2] . . . . .	387
18.157.2.2 <a href="#">WavefunctionUnion()</a> [2/2] . . . . .	387
18.157.3. <a href="#">Member Function Documentation</a> . . . . .	388
18.157.3.1 <a href="#">Ca_subset()</a> . . . . .	388
18.157.3.2 <a href="#">Cb_subset()</a> . . . . .	388
18.158. <a href="#">gefp.density.functional.XCFunctional Class Reference</a> . . . . .	389
18.158. Detailed Description . . . . .	390
18.158. <a href="#">Member Function Documentation</a> . . . . .	390
18.158.2. <a href="#">create()</a> . . . . .	390
<b>19 File Documentation</b>	<b>391</b>
19.1 <a href="#">include/oepdev_files.h File Reference</a> . . . . .	391
19.2 <a href="#">include/oepdev_options.h File Reference</a> . . . . .	391
19.3 <a href="#">main.cc File Reference</a> . . . . .	392
19.4 <a href="#">oepdev/lib3d/dmtp.h File Reference</a> . . . . .	393
19.5 <a href="#">oepdev/lib3d/esp.h File Reference</a> . . . . .	393
19.6 <a href="#">oepdev/libgefp/gefp.h File Reference</a> . . . . .	394
19.7 <a href="#">oepdev/libints/eri.h File Reference</a> . . . . .	395
19.8 <a href="#">oepdev/libints/recurr.h File Reference</a> . . . . .	396
19.9 <a href="#">oepdev/liboep/oep.h File Reference</a> . . . . .	397
19.10 <a href="#">oepdev/liboep/oep_gdf.h File Reference</a> . . . . .	398
19.11 <a href="#">oepdev/libpsi/integral.h File Reference</a> . . . . .	399
19.12 <a href="#">oepdev/libpsi/potential.h File Reference</a> . . . . .	399
19.13 <a href="#">oepdev/libsolver/solver.h File Reference</a> . . . . .	400
19.14 <a href="#">oepdev/libsolver/ti_data.h File Reference</a> . . . . .	401
19.15 <a href="#">oepdev/libtest/test.h File Reference</a> . . . . .	401
19.16 <a href="#">oepdev/libutil/basis_rotation.h File Reference</a> . . . . .	402
19.16.1 <a href="#">Theory</a> . . . . .	403
19.17 <a href="#">oepdev/libutil/cis.h File Reference</a> . . . . .	405

19.18	oepdev/libutil/davidson_liu.h File Reference . . . . .	406
19.19	oepdev/libutil/diis.h File Reference . . . . .	406
19.20	oepdev/libutil/gram_schmidt.h File Reference . . . . .	407
19.21	oepdev/libutil/integrals_iter.h File Reference . . . . .	407
19.22	oepdev/libutil/kabsch_superimposer.h File Reference . . . . .	408
19.23	oepdev/libutil/scf_perturb.h File Reference . . . . .	408
19.24	oepdev/libutil/unitary_optimizer.h File Reference . . . . .	409
19.25	oepdev/libutil/util.h File Reference . . . . .	410
19.26	oepdev/libutil/wavefunction_union.h File Reference . . . . .	412
<b>20</b>	<b>Example Documentation</b>	<b>415</b>
20.1	example_cphf.cc . . . . .	415
20.2	example_davidson_liu.cc . . . . .	415
20.3	example_integrals_iter.cc . . . . .	416
20.4	example_scf_perturb.cc . . . . .	417
	<b>Index</b>	<b>418</b>





# Chapter 1

## GEFP-OEP

### GEFP Project

The package is designed for quantum chemistry calculations. Currently, the implemented methods are:

- Density Matrix Functional Theory
- Density-Based decomposition of Mandado and Hermida-Ramon

The tutorial is under preparation.

### Installation

Installation prerequisites:

- Python 3.3 or newer
- NumPy module for Python, 1.16.3 or newer
- SciPy module for Python, 1.2.1 or newer
- Psi4 1.2.0 or newer
- OEP-Dev 1.0.3 or newer

To install the GEFP package type the following commands:

```
sudo python setup.py install
```



# Chapter 2

## License

**Copyright (c) 2018, Bartosz Błasiak**

All rights reserved.

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



# Chapter 3

## 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.

### 3.1 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:

$$\sum_f \left[ \left( \phi_i^A \phi_j^A || \phi_k^B \phi_l^B \right) \right] = \left( \phi_i^A | v_{kl}^B | \phi_j^A \right) \rightarrow \text{point charge or density fitting}$$

$$\sum_f \left[ \left( \phi_i^A \phi_j^B || \phi_k^B \phi_l^B \right) \right] = \left( \phi_i^A | v_{kl}^B | \phi_j^B \right) \rightarrow \text{density fitting,}$$

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.

### 3.2 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 losing 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.

### 3.3 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 programing conventions](#). The detailed discussion about using the OEPDev code can be found in [advanced usage section](#).





## Chapter 4

### 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.

#### 4.1 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  $\hat{v}^A$ .

##### 4.1.1 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
$\left(\mu \hat{v}^{A[\mu]} \sigma\right)$	$\sum_{l \in A} v_{\mu l}^A S_{l\sigma}$	$\sum_{\alpha \in A} q_{\alpha}^{A[\mu]} V_{\mu\sigma}^{(\alpha)}$
$\left(i \hat{v}^{A[i]} j\right)$	$\sum_{l \in A} v_{il}^A S_{lj}$	$\sum_{\alpha \in A} q_{\alpha}^{A[i]} V_{ij}^{(\alpha)}$
$\left(j \hat{v}^{A[i]} l\right)$	$\sum_{l \in A} S_{jl} v_{lk}^{A[i]} S_{kl}$	$\sum_{\alpha \in A} q_{\alpha}^{A[i]} V_{jl}^{(\alpha)}$

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

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

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

# Chapter 5

## 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.etal-S1` etc).

### 5.1 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:

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

### 5.2 Pauli Repulsion OEP's

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

### 5.2.1 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:

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

Matrix forms:

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

### 5.2.2 Second-order contribution in overlap matrix expansion.

To be added here!

## 5.3 Charge-Transfer Energy OEP's

To be added here!

## 5.4 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.

### 5.4.1 ET contributions.

3D forms:

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

Matrix forms:

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

#### 5.4.2 HT contributions.

Do be derived.

#### 5.4.3 CT contributions.

To be derived.

### 5.5 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 6

## 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.

### 6.1 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

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

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

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

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

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

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.

## 6.2 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

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

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

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

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

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 7

## Implemented Models

### 7.1 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)

### 7.2 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 8

## 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**.

### 8.1 Main routine and libraries

Oep-dev has only *one* source file in the plugin base directory, i.e., `main.cc`. This is the main driver routine that handles the functionality of the whole OEP testing platform: specifies options for Psi4 input file and implements test routines based on the options. 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.

### 8.2 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.

## 8.3 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.

## 8.4 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!

## 8.5 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`.

## 8.6 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-et al S1  ");
// Your code goes here
psi::timer_off("OEP      E(Paul) Murrell-et al 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.

## 8.7 Clean memory between independent jobs

If you use scratch disk space to store integrals, clean the scratch in between independent calculations. From C++ level invoke

```
#include "psi4/libpsio/psio.hpp"
// ...
psi::PSIOManager::shared_object()->psiclean();
```

whereas from the Python level use

```
import psi4
# ..
psi4.core.clean()
```

If the scratch space is not cleaned up before next independent task begins, certain computational routines might crash with `PSIOError` or continue without error, but produce wrong results.

## 8.8 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 9

## Advanced Usage

This section is addressed for advanced users.

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

### 9.1 Installation

#### 9.1.1 Preparing Psi4

OEPDev is a Psi4 plugin. It requires

- Psi4, at least 1.2 version (git commit `9d4a61c`). Has to be modified (see below)
- Eigen3, any newer version

#### Note

Before compiling, make sure EFP is enabled in `CMakeLists.txt` (now it is not used in OEPDev but maybe in the future it would).

Recently, Psi4 introduced API visibility management. Only certain Psi4 classes and functions are *exposed* in the `core.so` library, that is further linked to Psi4 plugin shared library. Due to this reason, not all Psi4 functionalities can be directly used from outside Psi4. In order to access local API of Psi4 (also used in the OEPDev code) slight modification of Psi4 code and concomitant rebuild is necessary.

In order to expose local API used by OEPDev and hidden within Psi4 1.2, two types of small modifications are necessary:

- M1: add `PSI_API` macro after required class or function declaration in header file
- M2: add `#include "psi4/pragma.h"` line at the include section of an appropriate header file

Modification M1 is obligatory for all affected files whereas modification M2 needs to be done only in headers that do not have "psi4/pragma.h" included explicitly or implicitly. The list of Psi4 header files along with the respective changes that need to be done are listed in the table below:

Psi4 Header File	Psi4 Class	Required Changes
libfunctional/superfunctional.h	Superfunctional	M1
libscf_solver/hf.h	HF	M1
libscf_solver/rhf.h	RHF	M1
libcubeprop/csg.h	CubicScalarGrid	M1
libmints/onebody.h	OneBodyAOInt	M1
libmints/potential.h	PotentialInt	M1
libmints/multipoles.h	MultipoleInt	M1
libmints/multipolesymmetry.h	MultipoleSymmetry	M1
libmints/fjt.h	Taylor_Fjt	M1
libmints/fjt.h	Fjt	M1
libmints/oeprop.h	OEProp	M1, M2
libmints/gshell.h	GaussianShell	M1, M2

To quickly apply these modifications, use the patch files stored in `misc/patch` directory. Please make sure to use a proper patch for a chosen Psi4 version.

### 9.1.2 Compilation

After all the above changes have been done in Psi4 (followed by its rebuild) compile the OEPDev code by running `compile` script. Make sure Eigen3 path is set to environment variable `EIGEN3_INCLUDE_DIR` (instructions will appear on the screen). After compilation is successful, run `ctest` to check if the code works fine.

#### Note

It may happen, that during code development there will be symbol lookup error when importing `oepdev.so` (in such case OEPDev compiles without error but Python cannot import the module `oepdev`). In such circumstance, probably there some local Psi4 feature that is needed in OEPDev is not exposed by `PSI_API` macro. To fix this, run `c++filt [name]` where `[name]` is the mangled undefined symbol. This will show you which Psi4 class or function is not exposed and requires `PSI_API` (change M1 and perhaps M2 too). Such change requires Psi4 rebuild and recompilation of OEPDev code. In any case, please contact me and report new undefined symbol ([blasiak.bartosz@gmail.com](mailto:blasiak.bartosz@gmail.com)).

## 9.2 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`.

### 9.2.1 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/CCTestTestfile.cmake.in` file.

### 9.2.2 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.

## 9.3 OEPDev Classes: Overview

### 9.3.1 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::OEPTyp` 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.

#### 9.3.1.1 OEPPotential

It is a container and computer class of OEP. Among others, the most important public method is `oepdev::OEPotential::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::OEPotential::oep` method, for instance. From protected attributes, each OE-Potential 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::OEPotential::build("ELECTROSTATIC ENERGY", wfn, options);
oep->compute();
oep->write_cube("V", "oep_cube_file");
```

So far, four OEPotential subclasses are implemented, from which `oepdev::ElectrostaticEnergyOEP` and `oepdev::RepulsionEnergyOEPotential` are fully operative, while the rest is under development.

#### 9.3.1.2 GeneralizedDensityFit

### 9.3.2 GEFP Module

#### 9.3.2.1 GenEffPar

#### 9.3.2.2 GenEffParFactory

#### 9.3.2.3 GenEffFrag

### 9.3.3 OEPDev Solver Module

#### 9.3.3.1 OEPDevSolver

## 9.4 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.

### 9.4.1 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);

    OEPType type_1 = {"Murrell-etal.S1", true, n1, mat_1};
    OEPType 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::OEPType`, 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  ");
}

```

#### 9.4.1.1 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 ESP-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-etal.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)!

#### 9.4.1.2 Abstract Base





# Chapter 10

## 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 11

## Module Index

### 11.1 Modules

Here is a list of all modules:

The Generalized One-Electron Potentials Library . . . . .	53
The OEPDev Solver Library . . . . .	54
The Generalized Effective Fragment Potentials Library . . . . .	55
The Integral Package Library . . . . .	57
The Three-Dimensional Vector Fields Library . . . . .	67
The Density Functional Theory Library . . . . .	70
The OEPDev Utilities . . . . .	71
The OEPDev Testing Platform Library . . . . .	86



# Chapter 12

## Namespace Index

### 12.1 Namespace List

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

<a href="#">gefp.basis.optimize</a>	87
<a href="#">gefp.density.dfi</a>	87
<a href="#">gefp.density.population</a>	88
<a href="#">oepdev</a>	
OEPDev module namespace	89
<a href="#">psi</a>	
Psi4 package namespace	98



# Chapter 13

## Hierarchical Index

### 13.1 Class Hierarchy

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

ABC

gefp.density.dfi.DFI	156
gefp.density.dfi.DFI_J	158
gefp.density.dfi.DFI_JK	158
oepdev::ABCD	104
gefp.density.dms.Aggregate	106
oepdev::AOIntegralsIterator	116
oepdev::AllAOIntegralsIterator_2	106
oepdev::AllAOIntegralsIterator_4	108
oepdev::CISData	134
oepdev::CPHF	137
CubicScalarGrid	
oepdev::CubePointsCollection3D	142
oepdev::DavidsonLiu	142
oepdev::CISComputer	127
oepdev::R_CISComputer	311
oepdev::R_CISComputer_Explicit	314
oepdev::R_CISComputer_Direct	311
oepdev::R_CISComputer_DL	312
oepdev::U_CISComputer	357
oepdev::U_CISComputer_Explicit	359
oepdev::U_CISComputer_DL	358
gefp.density.opdm.Density	146
gefp.density.functional.XCFunctional	389
gefp.density.functional.BBC1_XCFunctional	119
gefp.density.functional.BBC2_XCFunctional	120
gefp.density.functional.GU_XCFunctional	249
gefp.density.functional.HF_XCFunctional	251
gefp.density.functional.Interpolating_XCFunctional	254

gefp.density.functional.IDF1_XCFunctional . . . . .	252
gefp.density.functional.JKOnly_Interpolating_XCFunctional . . . . .	255
gefp.density.functional.MEDI_XCFunctional . . . . .	263
gefp.density.functional.V1_MEDI_XCFunctional . . . . .	379
gefp.density.functional.A_V1_MEDI_XCFunctional . . . . .	102
gefp.density.functional.V2_MEDI_XCFunctional . . . . .	380
gefp.density.functional.A_V2_MEDI_XCFunctional . . . . .	103
gefp.density.functional.P_V2_MEDI_XCFunctional . . . . .	292
gefp.density.functional.MBB_XCFunctional . . . . .	263
gefp.density.partitioning.DensityDecomposition . . . . .	150
gefp.basis.optimize.DFBasis . . . . .	154
gefp.basis.optimize.DFBasisOptimizer . . . . .	155
oepdev::DIISManager . . . . .	159
gefp.density.dmft.ElectronCorrelation . . . . .	195
gefp.density.dmft.DMFT . . . . .	160
gefp.density.dmft.DMFT_AO . . . . .	164
gefp.density.dmft.DMFT_NC . . . . .	165
gefp.density.dmft.DMFT_MO . . . . .	164
gefp.density.dmft.DMFT_PC . . . . .	165
gefp.density.dmft.DMFT_ProjD . . . . .	166
gefp.density.dmft.DMFT_ProjP . . . . .	167
enable_shared_from_this	
oepdev::DMTPole . . . . .	170
oepdev::CAMM . . . . .	121
oepdev::GenEffFrag . . . . .	216
oepdev::OEPDevSolver . . . . .	275
oepdev::ChargeTransferEnergySolver . . . . .	123
oepdev::EETCouplingSolver . . . . .	186
oepdev::ElectrostaticEnergySolver . . . . .	196
oepdev::RepulsionEnergySolver . . . . .	318
oepdev::OEPotential . . . . .	284
oepdev::ChargeTransferEnergyOEPotential . . . . .	122
oepdev::EETCouplingOEPotential . . . . .	185
oepdev::ElectrostaticEnergyOEPotential . . . . .	195
oepdev::RepulsionEnergyOEPotential . . . . .	317
gefp.core.driver.Entry . . . . .	201
oepdev::ESPSolver . . . . .	206
oepdev::Field3D . . . . .	211
oepdev::ElectrostaticPotential3D . . . . .	200
oepdev::OEPotential3D< T > . . . . .	289
oepdev::Fourier9 . . . . .	216
oepdev::GenEffPar . . . . .	222
oepdev::GenEffParFactory . . . . .	234
oepdev::EFP2_GEFactory . . . . .	193
oepdev::OEP_EFP2_GEFactory . . . . .	273
oepdev::PolarGEFactory . . . . .	302



oepdev::AbInitioPolarGEFactory . . . . .	105
oepdev::UnitaryTransformedMOPolarGEFactory . . . . .	377
oepdev::FFAbInitioPolarGEFactory . . . . .	210
oepdev::GeneralizedPolarGEFactory . . . . .	241
oepdev::NonUniformEFieldPolarGEFactory . . . . .	270
oepdev::LinearGradientNonUniformEFieldPolarGEFactory . . . . .	258
oepdev::LinearNonUniformEFieldPolarGEFactory . . . . .	260
oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory . . . . .	307
oepdev::QuadraticNonUniformEFieldPolarGEFactory . . . . .	309
oepdev::UniformEFieldPolarGEFactory . . . . .	360
oepdev::LinearUniformEFieldPolarGEFactory . . . . .	261
oepdev::QuadraticUniformEFieldPolarGEFactory . . . . .	310
oepdev::GeneralizedDensityFit . . . . .	238
oepdev::DoubleGeneralizedDensityFit . . . . .	182
oepdev::SingleGeneralizedDensityFit . . . . .	336
gefp.math.orthonorm.GrammSchmidt . . . . .	246
oepdev::GramSchmidt . . . . .	247
IntegralFactory	
oepdev::IntegralFactory . . . . .	253
oepdev::KabschSuperimposer . . . . .	256
gefp.density.population.Loc . . . . .	262
oepdev::MultipoleConvergence . . . . .	264
object	
gefp.core.utilities.UnitaryOptimizer . . . . .	367
gefp.core.utilities.UnitaryOptimizer_4.2 . . . . .	375
gefp.density.dmft.OEProp . . . . .	291
gefp.density.dmft.DMFT . . . . .	160
oepdev::OEType . . . . .	291
gefp.core.driver.PadeApproximant_2D . . . . .	293
oepdev::PerturbCharges . . . . .	294
oepdev::Points3DIterator::Point . . . . .	295
oepdev::Points3DIterator . . . . .	295
oepdev::CubePoints3DIterator . . . . .	140
oepdev::RandomPoints3DIterator . . . . .	315
oepdev::PointsCollection3D . . . . .	298
oepdev::CubePointsCollection3D . . . . .	142
oepdev::RandomPointsCollection3D . . . . .	316
PotentialInt	
oepdev::PotentialInt . . . . .	303
RHF	
oepdev::RHFPerturbed . . . . .	325
gefp.density.rvs.RVS . . . . .	327
gefp.density.dfi.SCF . . . . .	329
oepdev::ShellCombinationsIterator . . . . .	330
oepdev::AllAOShellCombinationsIterator_2 . . . . .	110
oepdev::AllAOShellCombinationsIterator_4 . . . . .	113

oepdev::GeneralizedPolarGEFactory::StatisticalSet	338
gefp.math.matrix.Superimposer	339
oepdev::test::Test	340
oepdev::TIData	343
TwoBodyAOInt	
oepdev::TwoBodyAOInt	353
oepdev::TwoElectronInt	354
oepdev::ERI_1_1	202
oepdev::ERI_2_2	203
oepdev::ERI_3_1	205
oepdev::UnitaryOptimizer	361
oepdev::UnitaryOptimizer_4_2	369
gefp.core.driver.UniversalSurface	378
Wavefunction	
oepdev::WavefunctionUnion	382
ABC	
gefp.basis.optimize.OEP	271
gefp.basis.optimize.OEP_FockLike	273
gefp.basis.optimize.OEP_CT	272
gefp.basis.optimize.OEP_Pauli	274
gefp.density.ci.CIWavefunction	135
gefp.density.ci.CIS_CIWavefunction	127
gefp.density.ci.HF_CIWavefunction	251
gefp.density.ci.SlaterDeterminant	337
gefp.density.ci.Reference_SlaterDeterminant	317
gefp.density.ci.Single_SlaterDeterminant	335
gefp.density.dmft.DMFT	160
gefp.density.dms.Computer	136
gefp.density.dms._DMS_SCF_Procedure	101
gefp.density.dms.Property_Computer	306
gefp.density.dms.OtherComputer	292
gefp.density.dms.SimpleComputer	335
gefp.density.dms.DMS	167
gefp.density.dms.Basic_DMS	118
gefp.density.dms.BasicD_DMS	119
gefp.density.dms.Field_DMS	215
gefp.density.dms.New2_DMS	269
gefp.density.dms.New_DMS	270
gefp.density.dms.DMSFit	169
gefp.density.dms._Global_Settings_DMSFit	101
gefp.density.dms.EFP_DMSFit	194
gefp.density.dms.ExternalField_EFP_DMSFit	209
gefp.density.dms.Rotation_DMSFit	327
gefp.density.dms.Translation_DMSFit	352
gefp.density.functional.XCFunctional	389
gefp.density.opdm.DensityProjection	154
gefp.density.opdm.Dset_DensityProjection	184

gefp.density.opdm.Pset_DensityProjection . . . . .	307
gefp.density.parameters.Guess . . . . .	250
gefp.density.parameters.Matrix_Guess . . . . .	262
gefp.density.parameters.NC_Guess . . . . .	268



# Chapter 14

## Class Index

### 14.1 Class List

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

<a href="#">gefp.density.dms._DMS_SCF_Procedure</a>	101
<a href="#">gefp.density.dms._Global_Settings_DMSFit</a>	101
<a href="#">gefp.density.functional.A_V1_MEDI_XCFunctional</a>	102
<a href="#">gefp.density.functional.A_V2_MEDI_XCFunctional</a>	103
<a href="#">oepdev::ABCD</a>	
Simple structure to hold the Fourier series expansion coefficients	104
<a href="#">oepdev::AbInitioPolarGEFactory</a>	
Polarization GEFP Factory from First Principles. Hartree-Fock Approximation	105
<a href="#">gefp.density.dms.Aggregate</a>	106
<a href="#">oepdev::AllAOIntegralsIterator_2</a>	
Loop over all possible ERI within a particular shell doublet	106
<a href="#">oepdev::AllAOIntegralsIterator_4</a>	
Loop over all possible ERI within a particular shell quartet	108
<a href="#">oepdev::AllAOShellCombinationsIterator_2</a>	
Loop over all possible ERI shells in a shell doublet	110
<a href="#">oepdev::AllAOShellCombinationsIterator_4</a>	
Loop over all possible ERI shells in a shell quartet	113
<a href="#">oepdev::AOIntegralsIterator</a>	
Iterator for AO Integrals. Abstract Base	116
<a href="#">gefp.density.dms.Basic_DMS</a>	118
<a href="#">gefp.density.dms.BasicD_DMS</a>	119
<a href="#">gefp.density.functional.BBC1_XCFunctional</a>	119
<a href="#">gefp.density.functional.BBC2_XCFunctional</a>	120
<a href="#">oepdev::CAMM</a>	
Cumulative Atomic Multipole Moments	121
<a href="#">oepdev::ChargeTransferEnergyOEPotential</a>	
Generalized One-Electron Potential for Charge-Transfer Interaction Energy	122
<a href="#">oepdev::ChargeTransferEnergySolver</a>	
Compute the Charge-Transfer interaction energy between unperturbed wave- functions	123

<a href="#">gefp.density.ci.CIS_CIWavefunction</a>	127
<a href="#">oepdev::CISComputer</a>	
<a href="#">CISComputer</a>	127
<a href="#">oepdev::CISData</a>	
Container to handle the CIS wavefunction parameters	134
<a href="#">gefp.density.ci.CIWavefunction</a>	135
<a href="#">gefp.density.dms.Computer</a>	136
<a href="#">oepdev::CPHF</a>	
CPHF solver class	137
<a href="#">oepdev::CubePoints3DIterator</a>	
Iterator over a collection of points in 3D space. g09 Cube-like order	140
<a href="#">oepdev::CubePointsCollection3D</a>	
G09 cube-like ordered collection of points in 3D space	142
<a href="#">oepdev::DavidsonLiu</a>	
Davidson-Liu diagonalization method	142
<a href="#">gefp.density.opdm.Density</a>	146
<a href="#">gefp.density.partitioning.DensityDecomposition</a>	150
<a href="#">gefp.density.opdm.DensityProjection</a>	154
<a href="#">gefp.basis.optimize.DFBasis</a>	154
<a href="#">gefp.basis.optimize.DFBasisOptimizer</a>	155
<a href="#">gefp.density.dfi.DFI</a>	156
<a href="#">gefp.density.dfi.DFI_J</a>	158
<a href="#">gefp.density.dfi.DFI_JK</a>	158
<a href="#">oepdev::DIISManager</a>	
DIIS manager	159
<a href="#">gefp.density.dmft.DMFT</a>	160
<a href="#">gefp.density.dmft.DMFT_AO</a>	164
<a href="#">gefp.density.dmft.DMFT_MO</a>	164
<a href="#">gefp.density.dmft.DMFT_NC</a>	165
<a href="#">gefp.density.dmft.DMFT_PC</a>	165
<a href="#">gefp.density.dmft.DMFT_ProjD</a>	166
<a href="#">gefp.density.dmft.DMFT_ProjP</a>	167
<a href="#">gefp.density.dms.DMS</a>	167
<a href="#">gefp.density.dms.DMSFit</a>	169
<a href="#">oepdev::DMTPole</a>	
Distributed Multipole Analysis Container and Computer. Abstract Base	170
<a href="#">oepdev::DoubleGeneralizedDensityFit</a>	
Generalized Density Fitting Scheme - Double Fit	182
<a href="#">gefp.density.opdm.Dset_DensityProjection</a>	184
<a href="#">oepdev::EETCouplingOEPotential</a>	
Generalized One-Electron Potential for EET coupling calculations	185
<a href="#">oepdev::EETCouplingSolver</a>	
Compute the EET coupling energy between unperturbed wavefunctions	186
<a href="#">oepdev::EFP2_GEFactory</a>	
EFP2 GEFP Factory	193
<a href="#">gefp.density.dms.EFP_DMSFit</a>	194
<a href="#">gefp.density.dmft.ElectronCorrelation</a>	195

<a href="#">oepdev::ElectrostaticEnergyOEPotential</a>	
Generalized One-Electron Potential for Electrostatic Energy . . . . .	195
<a href="#">oepdev::ElectrostaticEnergySolver</a>	
Compute the Coulombic interaction energy between unperturbed wavefunc- tions . . . . .	196
<a href="#">oepdev::ElectrostaticPotential3D</a>	
Electrostatic potential of a molecule . . . . .	200
<a href="#">gefp.core.driver.Entry</a> . . . . .	201
<a href="#">oepdev::ERI_1_1</a>	
2-centre ERI of the form $(a O(2) b)$ where $O(2) = 1/r_{12}$ . . . . .	202
<a href="#">oepdev::ERI_2_2</a>	
4-centre ERI of the form $(ab O(2) cd)$ where $O(2) = 1/r_{12}$ . . . . .	203
<a href="#">oepdev::ERI_3_1</a>	
4-centre ERI of the form $(abc O(2) d)$ where $O(2) = 1/r_{12}$ . . . . .	205
<a href="#">oepdev::ESPSolver</a>	
Charges from Electrostatic Potential (ESP). A solver-type class . . . . .	206
<a href="#">gefp.density.dms.ExternalField_EFP_DMSFit</a> . . . . .	209
<a href="#">oepdev::FFAbInitioPolarGEFactory</a>	
Polarization GEFP Factory from First Principles: Finite-Difference Model. Ar- bitrary level of theory . . . . .	210
<a href="#">oepdev::Field3D</a>	
General Vector Field in 3D Space. Abstract base . . . . .	211
<a href="#">gefp.density.dms.Field_DMS</a> . . . . .	215
<a href="#">oepdev::Fourier9</a>	
Simple structure to hold the Fourier series expansion coefficients for $N=4$ . . . .	216
<a href="#">oepdev::GenEffFrag</a>	
Generalized Effective Fragment. Container Class . . . . .	216
<a href="#">oepdev::GenEffPar</a>	
Generalized Effective Fragment Parameters. Container Class . . . . .	222
<a href="#">oepdev::GenEffParFactory</a>	
Generalized Effective Fragment Factory. Abstract Base . . . . .	234
<a href="#">oepdev::GeneralizedDensityFit</a>	
Generalized Density Fitting Scheme. Abstract Base . . . . .	238
<a href="#">oepdev::GeneralizedPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	241
<a href="#">gefp.math.orthonorm.GrammSchmidt</a> . . . . .	246
<a href="#">oepdev::GramSchmidt</a>	
Gram-Schmidt orthogonalization method . . . . .	247
<a href="#">gefp.density.functional.GU_XCFunctional</a> . . . . .	249
<a href="#">gefp.density.parameters.Guess</a> . . . . .	250
<a href="#">gefp.density.ci.HF_CIWavefunction</a> . . . . .	251
<a href="#">gefp.density.functional.HF_XCFunctional</a> . . . . .	251
<a href="#">gefp.density.functional.IDF1_XCFunctional</a> . . . . .	252
<a href="#">oepdev::IntegralFactory</a>	
Extended <a href="#">IntegralFactory</a> for computing integrals . . . . .	253
<a href="#">gefp.density.functional.Interpolating_XCFunctional</a> . . . . .	254
<a href="#">gefp.density.functional.JKOnly_Interpolating_XCFunctional</a> . . . . .	255

<a href="#">oepdev::KabschSuperimposer</a>	
Compute the Cartesian rotation matrix between two structures . . . . .	256
<a href="#">oepdev::LinearGradientNonUniformEFieldPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	258
<a href="#">oepdev::LinearNonUniformEFieldPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	260
<a href="#">oepdev::LinearUniformEFieldPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	261
<a href="#">gefp.density.population.Loc</a> . . . . .	262
<a href="#">gefp.density.parameters.Matrix_Guess</a> . . . . .	262
<a href="#">gefp.density.functional.MBB_XCFunctional</a> . . . . .	263
<a href="#">gefp.density.functional.MEDI_XCFunctional</a> . . . . .	263
<a href="#">oepdev::MultipoleConvergence</a>	
Multipole Convergence . . . . .	264
<a href="#">gefp.density.parameters.NC_Guess</a> . . . . .	268
<a href="#">gefp.density.dms.New2_DMS</a> . . . . .	269
<a href="#">gefp.density.dms.New_DMS</a> . . . . .	270
<a href="#">oepdev::NonUniformEFieldPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	270
<a href="#">gefp.basis.optimize.OEP</a> . . . . .	271
<a href="#">gefp.basis.optimize.OEP_CT</a> . . . . .	272
<a href="#">oepdev::OEP_EFP2_GEFactory</a>	
OEP-EFP2 GEFP Factory . . . . .	273
<a href="#">gefp.basis.optimize.OEP_FockLike</a> . . . . .	273
<a href="#">gefp.basis.optimize.OEP_Pauli</a> . . . . .	274
<a href="#">oepdev::OEPDevSolver</a>	
Solver of properties of molecular aggregates. Abstract base . . . . .	275
<a href="#">oepdev::OEPotential</a>	
Generalized One-Electron Potential: Abstract base . . . . .	284
<a href="#">oepdev::OEPotential3D&lt; T &gt;</a>	
Class template for OEP 3D fields . . . . .	289
<a href="#">gefp.density.dmft.OEProp</a> . . . . .	291
<a href="#">oepdev::OEPTyp</a>	
Container to handle the type of One-Electron Potentials . . . . .	291
<a href="#">gefp.density.dms.OtherComputer</a> . . . . .	292
<a href="#">gefp.density.functional.P_V2_MEDI_XCFunctional</a> . . . . .	292
<a href="#">gefp.core.driver.PadeApproximant_2D</a> . . . . .	293
<a href="#">oepdev::PerturbCharges</a>	
Structure to hold perturbing charges . . . . .	294
<a href="#">oepdev::Points3DIterator::Point</a> . . . . .	295
<a href="#">oepdev::Points3DIterator</a>	
Iterator over a collection of points in 3D space. Abstract base . . . . .	295
<a href="#">oepdev::PointsCollection3D</a>	
Collection of points in 3D space. Abstract base . . . . .	298
<a href="#">oepdev::PolarGEFactory</a>	
Polarization GEFP Factory. Abstract Base . . . . .	302
<a href="#">oepdev::PotentialInt</a>	
Computes potential integrals . . . . .	303



gefp.density.dms.Property_Computer	306
gefp.density.opdm.Pset_DensityProjection	307
oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory	
Polarization GEFP Factory with Least-Squares Parameterization	307
oepdev::QuadraticNonUniformEFieldPolarGEFactory	
Polarization GEFP Factory with Least-Squares Parameterization	309
oepdev::QuadraticUniformEFieldPolarGEFactory	
Polarization GEFP Factory with Least-Squares Parameterization	310
oepdev::R_CISComputer	311
oepdev::R_CISComputer_Direct	311
oepdev::R_CISComputer_DL	
CIS Computer with RHF reference: Davidson-Liu Solver	312
oepdev::R_CISComputer_Explicit	314
oepdev::RandomPoints3DIterator	
Iterator over a collection of points in 3D space. Random collection	315
oepdev::RandomPointsCollection3D	
Collection of random points in 3D space	316
gefp.density.ci.Reference_SlaterDeterminant	317
oepdev::RepulsionEnergyOEPotential	
Generalized One-Electron Potential for Pauli Repulsion Energy	317
oepdev::RepulsionEnergySolver	
Compute the Pauli-Repulsion interaction energy between unperturbed wave- functions	318
oepdev::RHPerturbed	
RHF theory under electrostatic perturbation	325
gefp.density.dms.Rotation_DMSFit	327
gefp.density.rvs.RVS	327
gefp.density.dfi.SCF	329
oepdev::ShellCombinationsIterator	
Iterator for Shell Combinations. Abstract Base	330
gefp.density.dms.SimpleComputer	335
gefp.density.ci.Single_SlaterDeterminant	335
oepdev::SingleGeneralizedDensityFit	
Generalized Density Fitting Scheme - Single Fit	336
gefp.density.ci.SlaterDeterminant	337
oepdev::GeneralizedPolarGEFactory::StatisticalSet	
A structure to handle statistical data	338
gefp.math.matrix.Superimposer	339
oepdev::test::Test	
Manages test routines	340
oepdev::TIData	
Transfer Integral EET Data	343
gefp.density.dms.Translation_DMSFit	352
oepdev::TwoBodyAOInt	353
oepdev::TwoElectronInt	
General Two Electron Integral	354
oepdev::U_CISComputer	357

<a href="#">oepdev::U_CISComputer_DL</a>	
CIS Computer with UHF reference: Davidson-Liu Solver . . . . .	358
<a href="#">oepdev::U_CISComputer_Explicit</a> . . . . .	359
<a href="#">oepdev::UniformEFieldPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Parameterization . . . . .	360
<a href="#">oepdev::UnitaryOptimizer</a>	
Find the optimim unitary matrix of quadratic matrix equation . . . . .	361
<a href="#">gefp.core.utilities.UnitaryOptimizer</a> . . . . .	367
<a href="#">oepdev::UnitaryOptimizer_4_2</a>	
Find the optimim unitary matrix for quartic-quadratic matrix equation with trace	369
<a href="#">gefp.core.utilities.UnitaryOptimizer_4_2</a> . . . . .	375
<a href="#">oepdev::UnitaryTransformedMOPolarGEFactory</a>	
Polarization GEFP Factory with Least-Squares Scaling of MO Space . . . . .	377
<a href="#">gefp.core.driver.UniversalSurface</a> . . . . .	378
<a href="#">gefp.density.functional.V1_MEDI_XCFunctional</a> . . . . .	379
<a href="#">gefp.density.functional.V2_MEDI_XCFunctional</a> . . . . .	380
<a href="#">oepdev::WavefunctionUnion</a>	
Union of two Wavefunction objects . . . . .	382
<a href="#">gefp.density.functional.XCFunctional</a> . . . . .	389

# Chapter 15

## File Index

### 15.1 File List

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

<a href="#">main.cc</a>	392
include/ <a href="#">oepdev_files.h</a>	391
include/ <a href="#">oepdev_options.h</a>	391
include/doxygen/ <b>oepdev_manual.h</b>	??
include/doxygen/ <b>oepdev_modules.h</b>	??
include/doxygen/ <b>oepdev_namespaces.h</b>	??
oepdev/lib3d/ <a href="#">dmtp.h</a>	393
oepdev/lib3d/ <a href="#">esp.h</a>	393
oepdev/lib3d/ <b>space3d.h</b>	??
oepdev/libgefp/ <a href="#">gefp.h</a>	394
oepdev/libints/ <a href="#">eri.h</a>	395
oepdev/libints/ <a href="#">recurr.h</a>	396
oepdev/liboep/ <a href="#">oep.h</a>	397
oepdev/liboep/ <a href="#">oep_gdf.h</a>	398
oepdev/libpsi/ <a href="#">integral.h</a>	399
oepdev/libpsi/ <a href="#">potential.h</a>	399
oepdev/libsolver/ <a href="#">solver.h</a>	400
oepdev/libsolver/ <a href="#">ti_data.h</a>	401
oepdev/libtest/ <a href="#">test.h</a>	401
oepdev/libutil/ <a href="#">basis_rotation.h</a>	402
oepdev/libutil/ <a href="#">cis.h</a>	405
oepdev/libutil/ <b>cphf.h</b>	??
oepdev/libutil/ <a href="#">davidson_liu.h</a>	406
oepdev/libutil/ <a href="#">diis.h</a>	406
oepdev/libutil/ <a href="#">gram_schmidt.h</a>	407
oepdev/libutil/ <a href="#">integrals_iter.h</a>	407
oepdev/libutil/ <a href="#">kabsch_superimposer.h</a>	408
oepdev/libutil/ <a href="#">scf_perturb.h</a>	408
oepdev/libutil/ <a href="#">unitary_optimizer.h</a>	409
oepdev/libutil/ <a href="#">util.h</a>	410

oepdev/libutil/[wavefunction\\_union.h](#) . . . . . 412

# Chapter 16

## Module Documentation

### 16.1 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.*

#### 16.1.1 Detailed Description

## 16.2 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.*
- class `oepdev::EETCouplingSolver`  
*Compute the EET coupling energy between unperturbed wavefunctions.*
- class `oepdev::TIData`  
*Transfer Integral EET Data.*

### 16.2.1 Detailed Description

## 16.3 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::EFP2_GEFactory`  
*EFP2 GEFP Factory.*
- class `oepdev::OEP_EFP2_GEFactory`  
*OEP-EFP2 GEFP Factory.*
- 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.*

### 16.3.1 Detailed Description



## 16.4 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  $(a|O(2)|b)$  where  $O(2) = 1/r^{12}$ .*
- class `oepdev::ERI_2_2`  
*4-centre ERI of the form  $(ab|O(2)|cd)$  where  $O(2) = 1/r^{12}$ .*
- class `oepdev::ERI_3_1`  
*4-centre ERI of the form  $(abc|O(2)|d)$  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.*

### 16.4.1 Detailed Description

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

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

### 16.4.2 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.

#### 16.4.2.1 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

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

### 16.4.3 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

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

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

### 16.4.4 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

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

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

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

### 16.4.5 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:

$$R_{0,0,M+1,j} = cR_{0,0,M,j+1} + MR_{0,0,M-1,j+1}$$

$$R_{0,L+1,M,j} = bR_{0,L,M,j+1} + LR_{0,L-1,M,j+1}$$

$$R_{N+1,L,M,j} = aR_{N,L,M,j+1} + NR_{N-1,L,M,j+1}$$

This scheme is implemented in OEPDev.

## 16.4.6 Function Documentation

### 16.4.6.1 d.N.n1.n2()

```
double oepdev::d.N.n1.n2 (
    int N,
    int n1,
    int n2,
    double PA,
    double PB,
    double aP )
```

#### 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

### 16.4.6.2 make\_mdh.D1\_coeff()

```
void oepdev::make_mdh.D1_coeff (
    int n1,
    double aPd,
    double * buffer )
```

#### 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](#)

### 16.4.6.3 make\_mdh\_D2\_coeff()

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

#### 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/Pa where Pa 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 n1+1 (0 to n1)</li> <li>• axis 2: dimension n2+1 (0 to n2)</li> <li>• axis 3: dimension n1+n2+1 (0 to n1+n2)</li> </ul>

See also

[D2.INDEX](#)

### 16.4.6.4 make\_mdh\_D2\_coeff\_explicit\_recursion()

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

## 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](#)16.4.6.5 `make_mdh_D3_coeff()`

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

## 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



## Parameters

<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 <math>n_1+1</math> (0 to <math>n_1</math>)</li> <li>• axis 2: dimension <math>n_2+1</math> (0 to <math>n_2</math>)</li> <li>• axis 3: dimension <math>n_3+1</math> (0 to <math>n_3</math>)</li> <li>• axis 4: dimension <math>n_1+n_2+n_3+1</math> (0 to <math>n_1+n_2+n_3</math>)</li> </ul>
---------------	---

## See also

[D3.INDEX](#)

16.4.6.6 `make_mdh_R_coeff()`

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

## Parameters

<i>N</i>	- increment in the summation of MDH series along <i>x</i> direction
<i>L</i>	- increment in the summation of MDH series along <i>y</i> direction
<i>M</i>	- increment in the summation of MDH series along <i>z</i> 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

**Parameters**

<i>buffer</i>	<div>- the McMurchie-Davidson 4-dimensional array (raveled to vector):<ul style="list-style-type: none"><li>• axis 0: dimension <math>N+1</math></li><li>• axis 1: dimension <math>L+1</math></li><li>• axis 2: dimension <math>M+1</math></li><li>• axis 3: dimension <math>N+L+M+1</math> (<math>j</math>-th element)</li></ul></div>
---------------	---

## 16.5 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::Camm`  
*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)*

### 16.5.1 Detailed Description

### 16.5.2 Function Documentation

#### 16.5.2.1 OEPotential3D() [1/2]

```
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 )
```

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

## 16.5.2.2 OEPotential3D() [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 )
```

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

## 16.6 The Density Functional Theory Library

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

## 16.7 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

- struct `oepdev::CISData`  
*Container to handle the CIS wavefunction parameters.*
- class `oepdev::CISComputer`  
*CISComputer.*
- class `oepdev::R_CISComputer`
- class `oepdev::U_CISComputer`
- class `oepdev::R_CISComputer_Explicit`
- class `oepdev::R_CISComputer_DL`  
*CIS Computer with RHF reference: Davidson-Liu Solver.*
- class `oepdev::R_CISComputer_Direct`
- class `oepdev::U_CISComputer_Explicit`
- class `oepdev::U_CISComputer_DL`  
*CIS Computer with UHF reference: Davidson-Liu Solver.*
- class `oepdev::CPHF`  
*CPHF solver class.*
- class `oepdev::DavidsonLiu`  
*Davidson-Liu diagonalization method.*
- class `oepdev::DIISManager`  
*DIIS manager.*
- class `oepdev::GramSchmidt`  
*Gram-Schmidt orthogonalization method.*
- 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.*

- class [oepdev::KabschSuperimposer](#)

*Compute the Cartesian rotation matrix between two structures.*

- 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)*

- `#define OEPDEV_AU_KcalPerMole 627.509`

*Energy converters.*

- `#define OEPDEV_AU_CMRec 219474.63`

- `#define OEPDEV_AU_EV 27.21138`



## 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

- PSI.API 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);`.*
- PSI.API `std::shared_ptr< SuperFunctional >` `oepdev::create_superfunctional` (std::string name, Options &options)  
*Set up DFT functional.*
- PSI.API `std::shared_ptr< Molecule >` `oepdev::extract_monomer` (std::shared\_ptr< const Molecule > molecule\_dimer, int id)  
*Extract molecule from dimer.*
- PSI.API double `oepdev::compute_distance` (psi::SharedVector v1, psi::SharedVector v2)  
*Compute distance between two points in nD space.*
- PSI.API `std::shared_ptr< Wavefunction >` `oepdev::solve_scf` (std::shared\_ptr< Molecule > molecule, std::shared\_ptr< BasisSet > primary, std::shared\_ptr< BasisSet > auxiliary, std::shared\_ptr< BasisSet > guess, std::shared\_ptr< SuperFunctional > functional, Options &options, std::shared\_ptr< PSIO > psio, bool compute\_mints=false)  
*Solve RHF-SCF equations for a given molecule in a given basis set.*
- PSI.API `std::shared_ptr< Wavefunction >` `oepdev::solve_scf_sad` (std::shared\_ptr< Molecule > molecule, std::shared\_ptr< BasisSet > primary, std::shared\_ptr< BasisSet > auxiliary, std::vector< std::shared\_ptr< BasisSet >> sad, std::vector< std::shared\_ptr< BasisSet >> sad\_fit, std::shared\_ptr< SuperFunctional > functional, Options &options, std::shared\_ptr< PSIO > psio, bool compute\_mints=false)  
*Solve RHF-SCF equations for a given molecule in a given basis set.*
- PSI.API double `oepdev::average_moment` (std::shared\_ptr< psi::Vector > moment)  
*Compute the scalar magnitude of multipole moment.*
- PSI.API `std::vector< std::shared_ptr< psi::Matrix >>` `oepdev::calculate_JK` (std::shared\_ptr< psi::Wavefunction > wfn, std::shared\_ptr< psi::Matrix > C)  
*Compute the Coulomb and exchange integral matrices in MO basis.*
- PSI.API `std::vector< std::shared_ptr< psi::Matrix >>` `oepdev::calculate_JK_r` (std::shared\_ptr< psi::Wavefunction > wfn, std::shared\_ptr< psi::IntegralTransform > tr, std::shared\_ptr< psi::Matrix > Dij)  
*Compute the Coulomb and exchange integral matrices in MO basis.*

- PSI\_API std::shared\_ptr< psi::Matrix > [oepdev::calculate\\_der\\_D](#) (std::shared\_ptr< psi::Wavefunction > wfn, std::shared\_ptr< psi::IntegralTransform > tr, std::shared\_ptr< psi::Matrix > C, std::vector< std::shared\_ptr< psi::Matrix >> A)  
*Compute the derivative of exchange-correlation energy wrt the density matrix in MO-A basis.*
- PSI\_API double [oepdev::calculate\\_e\\_xc](#) (std::shared\_ptr< psi::Wavefunction > wfn, std::shared\_ptr< psi::IntegralTransform > tr, std::shared\_ptr< psi::Matrix > f, std::shared\_ptr< psi::Matrix > C)  
*Compute the exchange-correlation energy from ERI in MO-SCF basis.*
- PSI\_API std::shared\_ptr< psi::Matrix > [oepdev::matrix\\_power\\_derivative](#) (std::shared\_ptr< psi::Matrix > A, double g, double step)  
*Compute the contracted derivative of power of a square and symmetric matrix.*
- std::shared\_ptr< psi::Matrix > [oepdev::calculate\\_DFI\\_Vel](#) (std::shared\_ptr< psi::IntegralFactory > f\_aabb, std::shared\_ptr< psi::IntegralFactory > f\_abab, std::shared\_ptr< psi::Matrix > d.b)  
*Compute the Effective DFI Potential Matrix Due To Electrons.*
- PSI\_API std::shared\_ptr< psi::Matrix > [oepdev::calculate\\_DFI\\_Vel\\_JK](#) (std::shared\_ptr< psi::IntegralFactory > f\_aabb, std::shared\_ptr< psi::IntegralFactory > f\_abab, std::shared\_ptr< psi::Matrix > d.b)  
*Compute the Effective DFI Coulomb+Exchange Potential Matrix Due To Electrons.*
- PSI\_API std::shared\_ptr< psi::Matrix > [oepdev::calculate\\_DFI\\_Vel\\_J](#) (std::shared\_ptr< psi::IntegralFactory > f\_aabb, std::shared\_ptr< psi::Matrix > d.b)  
*Compute the Effective DFI Coulomb Potential Matrix Due To Electrons.*

## Rotation of AO Space

### 16.7.1 Theory

The objective is to find the formulae for rotation matrices of the AO spaces as functions of the Cartesian 3 x 3 rotation matrices. It is obvious that  $p$ -type functions transform as a usual Cartesian vectors. However, higher angular momentum functions transform in a more complex way.

#### Problem

Define a vectorized AO space  $M$  of rank  $r > 1$  that is constructed from unique tensor components of fully symmetric  $r$ -th rank AO tensor populated in standard order,

$$M_{\{ab\dots k\}} = M_{ab\dots k} \quad \text{for } a \leq b \leq \dots \leq k$$

Given a general rotation of Cartesian tensors

$$M_{ab\dots k} = \sum_{a'b'\dots k'} M_{a'b'\dots k'} r_{a'a} r_{b'b} \dots r_{k'k}$$

find closed expressions for the rotation matrix in reduced composite AO space obeying

$$M_{[ab\dots k]} = \sum_{\{a'b'\dots k'\}} M_{\{a'b'\dots k'\}} R_{\{a'b'\dots k'\}, [ab\dots k]}$$

In the derivations below the following identity of first-order partitioning will be of use:

$$\sum_{ab} M_{ab} \hat{s}_{ab} = \sum_{\{ab\}} M_{\{ab\}} (\hat{s}_{ab} + \Delta_{ab} \hat{s}_{ba})$$

where

$$\Delta_{ab} \equiv 1 - \delta_{ab}$$

and the operator  $s$  of rank  $r$  acts as follows

$$s_{a'b'...k'}^{ab...k} \equiv \hat{s}_{a'b'...k'} \underbrace{\mathbf{r} \otimes \mathbf{r} \otimes \cdots \otimes \mathbf{r}}_r = r_{a'a} r_{b'b} \cdots r_{k'k}$$

### Rotation of 6D functions

The rotation of the 6D vector full tensor AO space of rank 2 and dimensions (3,3) is given by

$$M_{ab} = \sum_{a'b'} M_{a'b'} r_{a'b} r_{b'b}$$

Applying the identity of first-order partitioning directly leads to the formula for a reduced 6D tensor rotation of rank 1 and dimension (6),

$$M_{[ab]} = \sum_{\{a'b'\}} M_{\{a'b'\}} R_{\{a'b'\},[ab]}$$

where the 6 x 6 rotation matrix is given by

$$R_{\{a'b'\},[ab]} = r_{a'a} r_{b'b} + \Delta_{a'b'} r_{b'a} r_{a'b}$$

### Rotation of 10F functions

The rotation of the 10D vector full tensor AO space of rank 3 and dimensions (3,3,3) is given by

$$M_{abc} = \sum_{a'b'c'} M_{a'b'c'} r_{a'b} r_{b'b} r_{c'c}$$

First of all, notice that one can perform the following partitioning

$$\sum_a \sum_{b \neq a} \sum_{c \neq b \neq a} M_{abc} \hat{s}_{abc} = \sum_{\{abc\}} M_{\{abc\}} (\hat{s}_{abc} + \hat{s}_{acb} + \hat{s}_{bac} + \hat{s}_{bca} + \hat{s}_{cab} + \hat{s}_{cba})$$

Then, perform a partitioning of the triple sum,

$$\begin{aligned} \sum_{abc} M_{abc} \hat{s}_{abc} &= \sum_a \sum_{b \neq a} \sum_{c \neq b \neq a} M_{abc} \hat{s}_{abc} \\ &+ \sum_a \sum_{b \geq a} M_{abb} \hat{s}_{abb} + \sum_a \sum_{b < a} M_{abb} \hat{s}_{abb} \\ &+ \sum_a \sum_{b > a} M_{aba} \hat{s}_{aba} + \sum_a \sum_{b < a} M_{aba} \hat{s}_{aba} \\ &+ \sum_a \sum_{b > a} M_{bba} \hat{s}_{bba} + \sum_a \sum_{b < a} M_{bba} \hat{s}_{bba} \end{aligned}$$

Using the first-order partitioning theorem and interchanging the dummy indices one finds that

$$M_{[abc]} = \sum_{\{a'b'c'\}} M_{\{a'b'c'\}} R_{\{a'b'c'\},[abc]}$$

where the 10 x 10 rotation matrix is given by

$$\begin{aligned} R_{\{a'b'c'\},[abc]} = & \delta_{b'c'} \left( s_{a'b'b'}^{abc} + \Delta_{a'b'} \left\{ s_{b'a'b'}^{abc} + s_{b'b'a'}^{abc} \right\} \right) \\ & + \delta_{a'b'} \Delta_{b'c'} \left( s_{c'a'a'}^{abc} + s_{a'c'a'}^{abc} + s_{a'a'c'}^{abc} \right) \\ & + \Delta_{a'b'} \Delta_{b'c'} \left( s_{a'b'c}^{abc} + s_{a'c'b'}^{abc} + s_{b'a'c'}^{abc} + s_{b'c'a'}^{abc} + s_{c'a'b'}^{abc} + s_{c'b'a'}^{abc} \right) \end{aligned}$$

where

$$s_{a'b'c'}^{abc} \equiv \hat{s}_{a'b'c'} \mathbf{r} \otimes \mathbf{r} \otimes \mathbf{r} = r_{a'a} r_{b'b} r_{c'c}$$

- `psi::SharedMatrix oepdev::r6` (`psi::SharedMatrix r`)  
*Compute the 6 x 6 rotation matrix of the 6D orbitals.*
- `void oepdev::populate` (`double **R`, `double **r`, `std::vector< int > idx_am`, `const int &nam`)  
*Compute the 6 x 6 rotation matrix of the 6D orbitals.*
- `psi::SharedMatrix oepdev::ao_rotation_matrix` (`psi::SharedMatrix r`, `psi::SharedBasisSet b`)  
*Compute the full rotation matrix of AO orbital space.*

## 16.7.2 Detailed Description

## 16.7.3 Function Documentation

### 16.7.3.1 \_calculate\_DFI\_Vel()

```
std::shared_ptr< psi::Matrix > oepdev::_calculate_DFI_Vel (
    std::shared_ptr< psi::IntegralFactory > f_aabb,
    std::shared_ptr< psi::IntegralFactory > f_abab,
    std::shared_ptr< psi::Matrix > d_b )
```

Potential is felt by molecule A and induced by electrons in molecule B.

#### Parameters

<code>f_aabb</code>	- <a href="#">IntegralFactory</a> of type (AA BB)
<code>f_abab</code>	- <a href="#">IntegralFactory</a> of type (AB AB)
<code>d_b</code>	- one-particle density matrix in AO basis of B

**Returns**

- `V_el(B)` matrix in AO basis set of A

If `f_abab` is nullptr, then only Coulomb matrix is computed. Otherwise, also exchange contribution is computed.

**16.7.3.2 ao\_rotation\_matrix()**

```
psi::SharedMatrix oepdev::ao_rotation_matrix (
    psi::SharedMatrix r,
    psi::SharedBasisSet b )
```

**Parameters**

<i>r</i>	- Cartesian 3 x 3 rotation matrix
<i>b</i>	- Basis set

**16.7.3.3 average\_moment()**

```
PSI_API double oepdev::average_moment (
    std::shared_ptr< psi::Vector > 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.

#### 16.7.3.4 calculate\_der\_D()

```
PSI_API std::shared_ptr< psi::Matrix > oepdev::calculate_der_D (
    std::shared_ptr< psi::Wavefunction > wfn,
    std::shared_ptr< psi::IntegralTransform > tr,
    std::shared_ptr< psi::Matrix > C,
    std::vector< std::shared_ptr< psi::Matrix >> A )
```

Reads the existing MO ERI's.

##### Parameters

<i>wfn</i>	- Wavefunction object
<i>tr</i>	- IntegralTransform object
<i>C</i>	- Transformation matrix MO-B::MO-A (columns are MO-A basis)
<i>A</i>	- Vector of matrices $A^{(n)}$ - {bd}

##### Returns

- derivative matrix in MO-A basis

#### 16.7.3.5 calculate\_DFI\_Vel\_J()

```
PSI_API std::shared_ptr< psi::Matrix > oepdev::calculate_DFI_Vel_J (
    std::shared_ptr< psi::IntegralFactory > f_aabb,
    std::shared_ptr< psi::Matrix > d.b )
```

Potential is felt by molecule A and induced by electrons in molecule B.

##### Parameters

<i>f_aabb</i>	- <a href="#">IntegralFactory</a> of type (AA BB)
<i>d.b</i>	- one-particle density matrix in AO basis of B

##### Returns

-  $V_{el}(B)$  matrix in AO basis set of A

#### 16.7.3.6 calculate\_DFI\_Vel\_JK()

```
PSI_API std::shared_ptr< psi::Matrix > oepdev::calculate_DFI_Vel_JK (
    std::shared_ptr< psi::IntegralFactory > f_aabb,
```

```
std::shared_ptr< psi::IntegralFactory > f_abab,
std::shared_ptr< psi::Matrix > d_b )
```

Potential is felt by molecule A and induced by electrons in molecule B.

#### Parameters

<i>f_aabb</i>	- <a href="#">IntegralFactory</a> of type (AA BB)
<i>f_abab</i>	- <a href="#">IntegralFactory</a> of type (AB AB)
<i>d_b</i>	- one-particle density matrix in AO basis of B

#### Returns

-  $V_{el}(B)$  matrix in AO basis set of A

#### 16.7.3.7 calculate\_e\_xc()

```
PSI_API double oepdev::calculate_e_xc (
    std::shared_ptr< psi::Wavefunction > wfn,
    std::shared_ptr< psi::IntegralTransform > tr,
    std::shared_ptr< psi::Matrix > f,
    std::shared_ptr< psi::Matrix > C )
```

Reads the existing MO ERI's.

#### Parameters

<i>wfn</i>	- Wavefunction object
<i>tr</i>	- IntegralTransform object
<i>f</i>	- $f_{ij}$ matrix in MO-NEW basis
<i>C</i>	- Transformation matrix MO-SCF::MO-NEW (columns are MO-A basis)

#### Returns

- Exchange-correlation energy

#### 16.7.3.8 calculate\_JK()

```
PSI_API std::vector< std::shared_ptr< psi::Matrix > > oepdev::calculate_JK
(
    std::shared_ptr< psi::Wavefunction > wfn,
    std::shared_ptr< psi::Matrix > C )
```

Transforms the AO ERI's based on provided C matrix.

**Parameters**

<i>wfn</i>	- Wavefunction object
<i>C</i>	- molecular orbital coefficients (AO x MO)

**Returns**

- vector with J<sub>ij</sub> and K<sub>ij</sub> matrix

**16.7.3.9 calculate\_JK\_r()**

```
PSI_API std::vector< std::shared_ptr< psi::Matrix > > oepdev::calculate_JK_r
(
    std::shared_ptr< psi::Wavefunction > wfn,
    std::shared_ptr< psi::IntegralTransform > tr,
    std::shared_ptr< psi::Matrix > Dij )
```

Reads the existing MO ERI's.

**Parameters**

<i>wfn</i>	- Wavefunction object
<i>tr</i>	- IntegralTransform object
<i>D</i>	- density matrix in MO basis

**Returns**

- vector with J<sub>ij</sub> and K<sub>ij</sub> matrix

**16.7.3.10 compute\_distance()**

```
PSI_API double oepdev::compute_distance (
    psi::SharedVector v1,
    psi::SharedVector v2 )
```

**Parameters**

<i>v1</i>	- vector 1
<i>v2</i>	- vector 2



**Returns**

distance The vectors have to have the same length.

**16.7.3.11 create\_superfunctional()**

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

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](#).

**16.7.3.12 extract\_monomer()**

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

**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

### 16.7.3.13 matrix\_power\_derivative()

```
PSI_API std::shared_ptr< psi::Matrix > oepdev::matrix_power_derivative (
    std::shared_ptr< psi::Matrix > A,
    double g,
    double step )
```

The contracted matrix derivative is defined here as

$$\mathbf{D} = \frac{d\mathbf{A}^\gamma}{d\mathbf{A}} : \mathbb{I}$$

where  $\mathbb{I}$  is the identity matrix. The derivative, which is the fourth-rank tensor, is computed by the forward 2-centre finite difference formula,

$$f' = (f(h) - f(0)) / h$$

- if  $\gamma$  is non-integer, input matrix has to be positive-definite.

#### Parameters

<i>A</i>	- Matrix
<i>g</i>	- Power
<i>step</i>	- Differentiation step <i>h</i>

#### Returns

- Contracted derivative (matrix)

### 16.7.3.14 populate()

```
void oepdev::populate (
    double ** R,
    double ** r,
    std::vector< int > idx_am,
    const int & nam )
```

Compute the 10 x 10 rotation matrix of the 10F orbitals.

#### Parameters

<i>r</i>	- Cartesian 3 x 3 rotation matrix
----------	-----------------------------------

#### Returns

- 6 x 6 rotation matrix of the 6D orbitals

**Parameters**

<i>r</i>	- Cartesian 3 x 3 rotation matrix
----------	-----------------------------------

**Returns**

10 x 10 rotation matrix of the 10F orbitals

**16.7.3.15 r6()**

```
psi::SharedMatrix oepdev::r6 (  
    psi::SharedMatrix r )
```

Compute the 10 x 10 rotation matrix of the 10F orbitals.

**Parameters**

<i>r</i>	- Cartesian 3 x 3 rotation matrix
----------	-----------------------------------

**Returns**

6 x 6 rotation matrix of the 6D orbitals

**Parameters**

<i>r</i>	- Cartesian 3 x 3 rotation matrix
----------	-----------------------------------

**Returns**

10 x 10 rotation matrix of the 10F orbitals

**16.7.3.16 solve\_scf()**

```
PSI_API std::shared_ptr< Wavefunction > oepdev::solve_scf (  
    std::shared_ptr< Molecule > molecule,  
    std::shared_ptr< BasisSet > primary,  
    std::shared_ptr< BasisSet > auxiliary,  
    std::shared_ptr< BasisSet > guess,  
    std::shared_ptr< SuperFunctional > functional,  
    Options & options,  
    std::shared_ptr< PSIO > psio,  
    bool compute_mints = false )
```

**Parameters**

<i>molecule</i>	psi::SharedMolecule object with molecule
<i>primary</i>	basis set
<i>auxiliary</i>	basis set
<i>guess</i>	basis set
<i>functional</i>	DFT functional
<i>options</i>	psi::Options object
<i>psio</i>	psi::PSIO object
<i>compute_mints</i>	Compute integrals (write IWL TOC entry - necessary when transforming integrals)

**Returns**

psi::SharedWavefunction SCF wavefunction of the molecule

**16.7.3.17 solve\_scf\_sad()**

```
PSI_API std::shared_ptr< Wavefunction > oepdev::solve_scf_sad (
    std::shared_ptr< Molecule > molecule,
    std::shared_ptr< BasisSet > primary,
    std::shared_ptr< BasisSet > auxiliary,
    std::vector< std::shared_ptr< BasisSet >> sad,
    std::vector< std::shared_ptr< BasisSet >> sad_fit,
    std::shared_ptr< SuperFunctional > functional,
    Options & options,
    std::shared_ptr< PSIO > psio,
    bool compute_mints = false )
```

**Parameters**

<i>molecule</i>	psi::SharedMolecule object with molecule
<i>primary</i>	shared primary basis set
<i>auxiliary</i>	shared auxiliary basis set
<i>sad</i>	SAD basis set list
<i>sad_fit</i>	SAD DF fitting basis set list
<i>functional</i>	DFT functional
<i>options</i>	psi::Options object
<i>psio</i>	psi::PSIO object
<i>compute_mints</i>	Compute integrals (write IWL TOC entry - necessary when transforming integrals)

### Returns

psi::SharedWavefunction SCF wavefunction of the molecule

## 16.8 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.*

### 16.8.1 Detailed Description

# Chapter 17

## Namespace Documentation

### 17.1 `gefp.basis.optimize` Namespace Reference

#### Classes

- class [DFBasis](#)
- class [DFBasisOptimizer](#)
- class [OEP](#)
- class [OEP\\_CT](#)
- class [OEP\\_FockLike](#)
- class [OEP\\_Pauli](#)

#### Functions

- def **make\_bastempl** (templ, param)
- def **oeffitbasis** (mol, role='ORBITAL')
- def **removeComments** (string)

#### 17.1.1 Detailed Description

Auxiliary Basis Set Optimization Library.

The auxiliary basis sets for generalized density fitting (GDF) are here optimized.

### 17.2 `gefp.density.dfi` Namespace Reference

#### Classes

- class [DFI](#)
- class [DFI\\_J](#)

- class [DFI\\_JK](#)
- class [SCF](#)

## Variables

- int **MAX.NBF** = 128

### 17.2.1 Detailed Description

Demonstrates the use of Psi4 from Python level.

Useful notes:

- o Use psi4.core module for most of the work
- o Useful modules within psi4.core:
  - MintsHelper
  - Molecule
  - BasisSet
  - ExternalPotential
  - others
- o Psi4 defines its own matrix type (psi4.core.Matrix).  
Extracting numpy.array is easy:
 

```
numpy_array = numpy.asarray(psi4_matrix)
```

 Creating Psi4 matrix from array is also easy:
 

```
psi4_matrix = psi4.core.Matrix.from_array(numpy_array)
```
- o To compute 1-el potential matrix for a set of charges  
use ExternalPotential (charge positions are to be provided in Angstroms)  
unless charges are just nuclei within the basis set (in this case use of ao\_potential  
of MintsHelper is easier).
- o ao\_potential method of MintsHelper is limited only for nuclei within the same basis  
(the nuclei are taken from the first basis set axis, for example:
 

```
mints = MintsHelper(basis_X)
mints.ao_potential()          -> nuclei taken from basis of mints object (b
mints.ao_potential(basis_1, basis_2) -> nuclei taken from basis_1
```
- o Psi4 has efficient and easy to use method of defining fragments within a molecule (u  
Defining ghost atoms and extracting fragment i in the multimer-centred basis set is  
(method extract\_subsets(...) of psi4.core.Molecule)

## 17.3 gefp.density.population Namespace Reference

### Classes

- class [Loc](#)

### Functions

- def [atomic\\_charges](#) (wfn, kappa=0.0)



### 17.3.1 Detailed Description

Module for population analyses.

Bartosz B\_lasiak, Gundelfingen, September 2019

Notes:

Copied from my QC Workshop.

Reference: [https://github.com/globulion/qc-workshop/tree/master/tutor/project\\_1#popul](https://github.com/globulion/qc-workshop/tree/master/tutor/project_1#popul)

### 17.3.2 Function Documentation

#### 17.3.2.1 atomic\_charges()

```
def gefp.density.population.atomic_charges (
    wfn,
    kappa = 0.0 )
```

Compute atomic partial charges as a function of kappa parameter.

Input:

wfn - psi4.core.Wavefunction object  
kappa - parameter in the interval [0, 1]

Returns:

numpy.ndarray of shape (wfn.molecule().natom(), ) with partial charges [A.U.]

Notes:

- o kappa = 0 corresponds to Mulliken charges
- o kappa = 1/2 corresponds to Lowdin charges

## 17.4 oepdev Namespace Reference

OEPDev module namespace.

### Classes

- struct [ABCD](#)

*Simple structure to hold the Fourier series expansion coefficients.*

- class [AbInitioPolarGEFactory](#)

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

- class [AllAOIntegralsIterator\\_2](#)

*Loop over all possible ERI within a particular shell doublet.*

- class [AllAOIntegralsIterator\\_4](#)

*Loop over all possible ERI within a particular shell quartet.*

- class [AllAOShellCombinationsIterator\\_2](#)  
*Loop over all possible ERI shells in a shell doublet.*
- class [AllAOShellCombinationsIterator\\_4](#)  
*Loop over all possible ERI shells in a shell quartet.*
- class [AOIntegralsIterator](#)  
*Iterator for AO Integrals. Abstract Base.*
- class [CAMM](#)  
*Cumulative Atomic Multipole Moments.*
- class [ChargeTransferEnergyOEPotential](#)  
*Generalized One-Electron Potential for Charge-Transfer Interaction Energy.*
- class [ChargeTransferEnergySolver](#)  
*Compute the Charge-Transfer interaction energy between unperturbed wavefunctions.*
- class [CISComputer](#)  
*CISComputer.*
- struct [CISData](#)  
*Container to handle the CIS wavefunction parameters.*
- class [CPHF](#)  
*CPHF solver class.*
- class [CubePoints3DIterator](#)  
*Iterator over a collection of points in 3D space. g09 Cube-like order.*
- class [CubePointsCollection3D](#)  
*G09 cube-like ordered collection of points in 3D space.*
- class [DavidsonLiu](#)  
*Davidson-Liu diagonalization method.*
- class [DIISManager](#)  
*DIIS manager.*
- class [DMTPole](#)  
*Distributed Multipole Analysis Container and Computer. Abstract Base.*
- class [DoubleGeneralizedDensityFit](#)  
*Generalized Density Fitting Scheme - Double Fit.*
- class [EETCouplingOEPotential](#)  
*Generalized One-Electron Potential for EET coupling calculations.*
- class [EETCouplingSolver](#)  
*Compute the EET coupling energy between unperturbed wavefunctions.*
- class [EFP2.GEFPFactory](#)  
*EFP2 GEFP Factory.*
- class [ElectrostaticEnergyOEPotential](#)  
*Generalized One-Electron Potential for Electrostatic Energy.*
- class [ElectrostaticEnergySolver](#)  
*Compute the Coulombic interaction energy between unperturbed wavefunctions.*

- class [ElectrostaticPotential3D](#)  
*Electrostatic potential of a molecule.*
- class [ERI\\_1\\_1](#)  
*2-centre ERI of the form  $(a|O(2)|b)$  where  $O(2) = 1/r_{12}$ .*
- class [ERI\\_2\\_2](#)  
*4-centre ERI of the form  $(ab|O(2)|cd)$  where  $O(2) = 1/r_{12}$ .*
- class [ERI\\_3\\_1](#)  
*4-centre ERI of the form  $(abc|O(2)|d)$  where  $O(2) = 1/r_{12}$ .*
- class [ESPSolver](#)  
*Charges from Electrostatic Potential (ESP). A solver-type class.*
- class [FFAbInitioPolarGEFactory](#)  
*Polarization GEFP Factory from First Principles: Finite-Difference Model. Arbitrary level of theory.*
- class [Field3D](#)  
*General Vector Field in 3D Space. Abstract base.*
- struct [Fourier9](#)  
*Simple structure to hold the Fourier series expansion coefficients for  $N=4$ .*
- class [GenEffFrag](#)  
*Generalized Effective Fragment. Container Class.*
- class [GenEffPar](#)  
*Generalized Effective Fragment Parameters. Container Class.*
- class [GenEffParFactory](#)  
*Generalized Effective Fragment Factory. Abstract Base.*
- class [GeneralizedDensityFit](#)  
*Generalized Density Fitting Scheme. Abstract Base.*
- class [GeneralizedPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [GramSchmidt](#)  
*Gram-Schmidt orthogonalization method.*
- class [IntegralFactory](#)  
*Extended [IntegralFactory](#) for computing integrals.*
- class [KabschSuperimposer](#)  
*Compute the Cartesian rotation matrix between two structures.*
- class [LinearGradientNonUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [LinearNonUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [LinearUniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- class [MultipoleConvergence](#)

- Multipole Convergence.*

  - class [NonUniformEFieldPolarGEFactory](#)

*Polarization GEFP Factory with Least-Squares Parameterization.*
  - class [OEP\\_EFP2\\_GEFactory](#)

*OEP-EFP2 GEFP Factory.*
  - class [OEPDevSolver](#)

*Solver of properties of molecular aggregates. Abstract base.*
  - class [OEPotential](#)

*Generalized One-Electron Potential: Abstract base.*
  - class [OEPotential3D](#)

*Class template for OEP 3D fields.*
  - struct [OEPTyp](#)

*Container to handle the type of One-Electron Potentials.*
  - struct [PerturbCharges](#)

*Structure to hold perturbing charges.*
  - class [Points3DIterator](#)

*Iterator over a collection of points in 3D space. Abstract base.*
  - class [PointsCollection3D](#)

*Collection of points in 3D space. Abstract base.*
  - class [PolarGEFactory](#)

*Polarization GEFP Factory. Abstract Base.*
  - class [PotentialInt](#)

*Computes potential integrals.*
  - class [QuadraticGradientNonUniformEFieldPolarGEFactory](#)

*Polarization GEFP Factory with Least-Squares Parameterization.*
  - class [QuadraticNonUniformEFieldPolarGEFactory](#)

*Polarization GEFP Factory with Least-Squares Parameterization.*
  - class [QuadraticUniformEFieldPolarGEFactory](#)

*Polarization GEFP Factory with Least-Squares Parameterization.*
  - class [R\\_CISComputer](#)
  - class [R\\_CISComputer\\_Direct](#)
  - class [R\\_CISComputer\\_DL](#)

*CIS Computer with RHF reference: Davidson-Liu Solver.*
  - class [R\\_CISComputer\\_Explicit](#)
  - class [RandomPoints3DIterator](#)

*Iterator over a collection of points in 3D space. Random collection.*
  - class [RandomPointsCollection3D](#)

*Collection of random points in 3D space.*
  - class [RepulsionEnergyOEPotential](#)

*Generalized One-Electron Potential for Pauli Repulsion Energy.*

- class [RepulsionEnergySolver](#)  
*Compute the Pauli-Repulsion interaction energy between unperturbed wavefunctions.*
- class [RHPerturbed](#)  
*RHF theory under electrostatic perturbation.*
- class [ShellCombinationsIterator](#)  
*Iterator for Shell Combinations. Abstract Base.*
- class [SingleGeneralizedDensityFit](#)  
*Generalized Density Fitting Scheme - Single Fit.*
- class [TIData](#)  
*Transfer Integral EET Data.*
- class [TwoBodyAOInt](#)
- class [TwoElectronInt](#)  
*General Two Electron Integral.*
- class [U\\_CISComputer](#)
- class [U\\_CISComputer\\_DL](#)  
*CIS Computer with UHF reference: Davidson-Liu Solver.*
- class [U\\_CISComputer\\_Explicit](#)
- class [UniformEFieldPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Parameterization.*
- 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 [UnitaryTransformedMOPolarGEFactory](#)  
*Polarization GEFP Factory with Least-Squares Scaling of MO Space.*
- 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 **SharedLocalizer** = std::shared\_ptr< [Localizer](#) >
- using **SharedCISData** = std::shared\_ptr< [CISData](#) >
- using **SharedWavefunctionUnion** = std::shared\_ptr< [WavefunctionUnion](#) >
- using **SharedOEPotential** = std::shared\_ptr< [OEPotential](#) >
- using **SharedDMTPConvergence** = std::shared\_ptr< [oepdev::MultipoleConvergence](#) >

- using **SharedMolecule** = std::shared\_ptr< psi::Molecule >
- using **SharedMOSpace** = std::shared\_ptr< psi::MOSpace >
- using **SharedMOSpaceVector** = std::vector< std::shared\_ptr< psi::MOSpace > >
- using **SharedIntegralTransform** = std::shared\_ptr< psi::IntegralTransform >
- 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 **SharedSuperFunctional** = std::shared\_ptr< SuperFunctional >

## Functions

- 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.*
- psi::SharedMatrix **r10** (psi::SharedMatrix r3)
- constexpr std::complex< double > **operator""\_i** (unsigned long long d)
- constexpr std::complex< double > **operator""\_i** (long double d)
- PSI.API void [preamble](#) (void)  
*Print preamble for module OEPDEV.*
- PSI.API std::shared\_ptr< SuperFunctional > [create\\_superfunctional](#) (std::string name, Options &options)  
*Set up DFT functional.*
- PSI.API std::shared\_ptr< Molecule > [extract\\_monomer](#) (std::shared\_ptr< const Molecule > molecule\_dimer, int id)

*Extract molecule from dimer.*

- PSI.API double [compute\\_distance](#) (psi::SharedVector v1, psi::SharedVector v2)

*Compute distance between two points in nD space.*

- PSI.API std::shared\_ptr< Wavefunction > [solve\\_scf](#) (std::shared\_ptr< Molecule > molecule, std::shared\_ptr< BasisSet > primary, std::shared\_ptr< BasisSet > auxiliary, std::shared\_ptr< BasisSet > guess, std::shared\_ptr< SuperFunctional > functional, Options &options, std::shared\_ptr< PSIO > psio, bool compute\_mints=false)

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

- PSI.API std::shared\_ptr< Wavefunction > [solve\\_scf\\_sad](#) (std::shared\_ptr< Molecule > molecule, std::shared\_ptr< BasisSet > primary, std::shared\_ptr< BasisSet > auxiliary, std::vector< std::shared\_ptr< BasisSet >> sad, std::vector< std::shared\_ptr< BasisSet >> sad\_fit, std::shared\_ptr< SuperFunctional > functional, Options &options, std::shared\_ptr< PSIO > psio, bool compute\_mints=false)

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

- PSI.API double [average\\_moment](#) (std::shared\_ptr< psi::Vector > moment)

*Compute the scalar magnitude of multipole moment.*

- PSI.API std::vector< std::shared\_ptr< psi::Matrix > > [calculate\\_JK](#) (std::shared\_ptr< psi::Wavefunction > wfn, std::shared\_ptr< psi::Matrix > C)

*Compute the Coulomb and exchange integral matrices in MO basis.*

- PSI.API std::vector< std::shared\_ptr< psi::Matrix > > [calculate\\_JK\\_r](#) (std::shared\_ptr< psi::Wavefunction > wfn, std::shared\_ptr< psi::IntegralTransform > tr, std::shared\_ptr< psi::Matrix > Dij)

*Compute the Coulomb and exchange integral matrices in MO basis.*

- PSI.API std::shared\_ptr< psi::Matrix > [calculate\\_der\\_D](#) (std::shared\_ptr< psi::Wavefunction > wfn, std::shared\_ptr< psi::IntegralTransform > tr, std::shared\_ptr< psi::Matrix > C, std::vector< std::shared\_ptr< psi::Matrix >> A)

*Compute the derivative of exchange-correlation energy wrt the density matrix in MO-A basis.*

- PSI.API double [calculate\\_e\\_xc](#) (std::shared\_ptr< psi::Wavefunction > wfn, std::shared\_ptr< psi::IntegralTransform > tr, std::shared\_ptr< psi::Matrix > f, std::shared\_ptr< psi::Matrix > C)

*Compute the exchange-correlation energy from ERI in MO-SCF basis.*

- PSI.API std::shared\_ptr< psi::Matrix > [matrix\\_power\\_derivative](#) (std::shared\_ptr< psi::Matrix > A, double g, double step)

*Compute the contracted derivative of power of a square and symmetric matrix.*

- std::shared\_ptr< psi::Matrix > [\\_calculate\\_DFI\\_Vel](#) (std::shared\_ptr< psi::IntegralFactory > f\_aabb, std::shared\_ptr< psi::IntegralFactory > f\_abab, std::shared\_ptr< psi::Matrix > d\_b)

*Compute the Effective DFI Potential Matrix Due To Electrons.*

- PSI.API std::shared\_ptr< psi::Matrix > [calculate\\_DFI\\_Vel\\_JK](#) (std::shared\_ptr< psi::IntegralFactory > f\_aabb, std::shared\_ptr< psi::IntegralFactory > f\_abab, std::shared\_ptr< psi::Matrix > d\_b)

*Compute the Effective DFI Coulomb+Exchange Potential Matrix Due To Electrons.*

- PSI.API std::shared\_ptr< psi::Matrix > [calculate\\_DFI\\_Vel\\_J](#) (std::shared\_ptr< psi::IntegralFactory > f\_aabb, std::shared\_ptr< psi::Matrix > d\_b)

*Compute the Effective DFI Coulomb Potential Matrix Due To Electrons.*

- `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);`*

## Rotation of AO Space

### 17.4.1 Theory

The objective is to find the formulae for rotation matrices of the AO spaces as functions of the Cartesian  $3 \times 3$  rotation matrices. It is obvious that  $p$ -type functions transform as a usual Cartesian vectors. However, higher angular momentum functions transform in a more complex way.

#### Problem

Define a vectorized AO space  $M$  of rank  $r > 1$  that is constructed from unique tensor components of fully symmetric  $r$ -th rank AO tensor populated in standard order,

$$M_{\{ab\dots k\}} = M_{ab\dots k} \quad \text{for } a \leq b \leq \dots \leq k$$

Given a general rotation of Cartesian tensors

$$M_{ab\dots k} = \sum_{a'b'\dots k'} M_{a'b'\dots k'} r_{a'a} r_{b'b} \dots r_{k'k}$$

find closed expressions for the rotation matrix in reduced composite AO space obeying

$$M_{[ab\dots k]} = \sum_{\{a'b'\dots k'\}} M_{\{a'b'\dots k'\}} R_{\{a'b'\dots k'\}, [ab\dots k]}$$

In the derivations below the following identity of first-order partitioning will be of use:

$$\sum_{ab} M_{ab} \hat{s}_{ab} = \sum_{\{ab\}} M_{\{ab\}} (\hat{s}_{ab} + \Delta_{ab} \hat{s}_{ba})$$

where

$$\Delta_{ab} \equiv 1 - \delta_{ab}$$

and the operator  $s$  of rank  $r$  acts as follows

$$s_{a'b'\dots k'}^{ab\dots k} \equiv \hat{s}_{a'b'\dots k'} \underbrace{\mathbf{r} \otimes \mathbf{r} \otimes \dots \otimes \mathbf{r}}_r = r_{a'a} r_{b'b} \dots r_{k'k}$$

#### Rotation of 6D functions

The rotation of the 6D vector full tensor AO space of rank 2 and dimensions (3,3) is given by

$$M_{ab} = \sum_{a'b'} M_{a'b'} r_{a'a} r_{b'b}$$



Applying the identity of first-order partitioning directly leads to the formula for a reduced 6D tensor rotation of rank 1 and dimension (6),

$$M_{[ab]} = \sum_{\{a'b'\}} M_{\{a'b'\}} R_{\{a'b'\},[ab]}$$

where the 6 x 6 rotation matrix is given by

$$R_{\{a'b'\},[ab]} = r_{a'a} r_{b'b} + \Delta_{a'b'} r_{b'a} r_{a'b}$$

#### Rotation of 10F functions

The rotation of the 10D vector full tensor AO space of rank 3 and dimensions (3,3,3) is given by

$$M_{abc} = \sum_{a'b'c'} M_{a'b'c'} r_{a'b} r_{b'b} r_{c'c}$$

First of all, notice that one can perform the following partitioning

$$\sum_a \sum_{b \neq a} \sum_{c \neq b \neq a} M_{abc} \hat{s}_{abc} = \sum_{\{abc\}} M_{\{abc\}} (\hat{s}_{abc} + \hat{s}_{acb} + \hat{s}_{bac} + \hat{s}_{bca} + \hat{s}_{cab} + \hat{s}_{cba})$$

Then, perform a partitioning of the triple sum,

$$\begin{aligned} \sum_{abc} M_{abc} \hat{s}_{abc} &= \sum_a \sum_{b \neq a} \sum_{c \neq b \neq a} M_{abc} \hat{s}_{abc} \\ &+ \sum_a \sum_{b \geq a} M_{abb} \hat{s}_{abb} + \sum_a \sum_{b < a} M_{abb} \hat{s}_{abb} \\ &+ \sum_a \sum_{b > a} M_{aba} \hat{s}_{aba} + \sum_a \sum_{b < a} M_{aba} \hat{s}_{aba} \\ &+ \sum_a \sum_{b > a} M_{bba} \hat{s}_{bba} + \sum_a \sum_{b < a} M_{bba} \hat{s}_{bba} \end{aligned}$$

Using the first-order partitioning theorem and interchanging the dummy indices one finds that

$$M_{[abc]} = \sum_{\{a'b'c'\}} M_{\{a'b'c'\}} R_{\{a'b'c'\},[abc]}$$

where the 10 x 10 rotation matrix is given by

$$\begin{aligned} R_{\{a'b'c'\},[abc]} &= \delta_{b'c'} \left( s_{a'b'b'}^{abc} + \Delta_{a'b'} \left\{ s_{b'a'b'}^{abc} + s_{b'b'a'}^{abc} \right\} \right) \\ &+ \delta_{a'b'} \Delta_{b'c'} \left( s_{c'a'a'}^{abc} + s_{a'c'a'}^{abc} + s_{a'a'c'}^{abc} \right) \\ &+ \Delta_{a'b'} \Delta_{b'c'} \left( s_{a'b'b'}^{abc} + s_{a'c'b'}^{abc} + s_{b'a'c'}^{abc} + s_{b'b'c'a'}^{abc} + s_{c'a'b'}^{abc} + s_{c'b'a'}^{abc} \right) \end{aligned}$$

where

$$s_{a'b'c'}^{abc} \equiv \hat{s}_{a'b'c'} \mathbf{r} \otimes \mathbf{r} \otimes \mathbf{r} = r_{a'a} r_{b'b} r_{c'c}$$

- `psi::SharedMatrix r6` (`psi::SharedMatrix r`)  
Compute the 6 x 6 rotation matrix of the 6D orbitals.
- `void populate` (`double **R`, `double **r`, `std::vector< int > idx_am`, `const int &nam`)  
Compute the 6 x 6 rotation matrix of the 6D orbitals.
- `psi::SharedMatrix ao_rotation_matrix` (`psi::SharedMatrix r`, `psi::SharedBasisSet b`)  
Compute the full rotation matrix of AO orbital space.

### 17.4.2 Detailed Description

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

## 17.5 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

- PSI\_API int [read\\_options](#) (std::string name, Options &options)  
*Options for the OEPDev plugin.*
- void **export\_dmt** (py::module &)
- void **export\_cphf** (py::module &)
- void **export\_solver** (py::module &)
- void **export\_util** (py::module &)
- void **export\_oep** (py::module &)
- void **export\_gefp** (py::module &)
- PSI\_API SharedWavefunction [oepdev](#) (SharedWavefunction ref\_wfn, Options &options)  
*Main routine of the OEPDev plugin.*
- **PYBIND11\_MODULE** ([oepdev](#), m)

### 17.5.1 Detailed Description

Contains all Psi4 functionalities.

### 17.5.2 Function Documentation

### 17.5.2.1 oepdev()

```
PSI_API SharedWavefunction psi::oepdev (
    SharedWavefunction ref_wfn,
    Options & options )
```

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)

### 17.5.2.2 read\_options()

```
PSI_API int psi::read_options (
    std::string name,
    Options & options )
```

#### Parameters

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

#### Returns

true

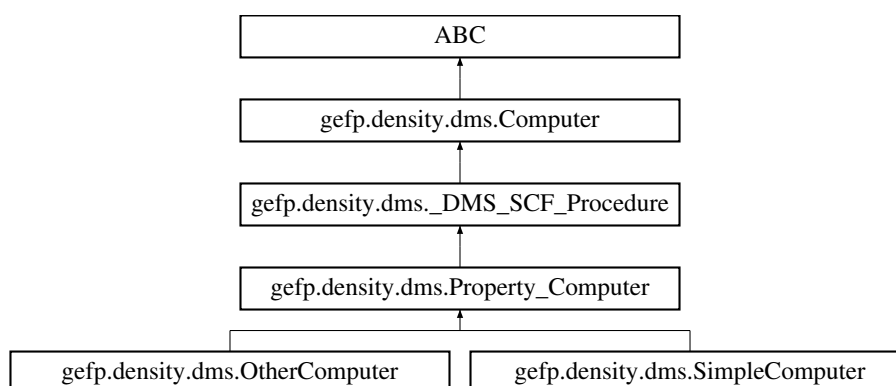


# Chapter 18

## Class Documentation

### 18.1 `gefp.density.dms._DMS_SCF_Procedure` Class Reference

Inheritance diagram for `gefp.density.dms._DMS_SCF_Procedure`:



#### Public Member Functions

- `def __init__ (self, aggregate, fragments)`

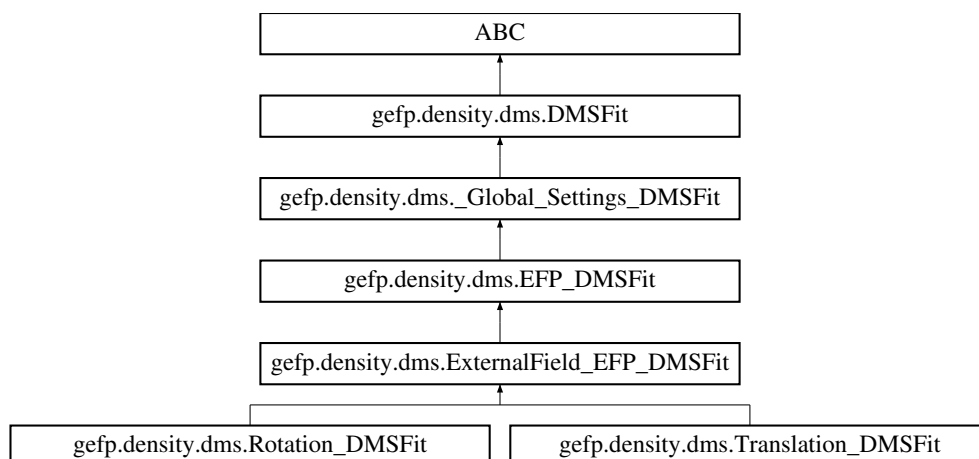
#### Additional Inherited Members

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

- `gefp/gefp/density/dms.py`

### 18.2 `gefp.density.dms._Global_Settings_DMSFit` Class Reference

Inheritance diagram for `gefp.density.dms._Global_Settings_DMSFit`:



## Public Member Functions

- `def __init__ (self, mol, method, nsamples, dms_types, order_type, use_iterative_model, use_external_field_model)`

## Additional Inherited Members

### 18.2.1 Detailed Description

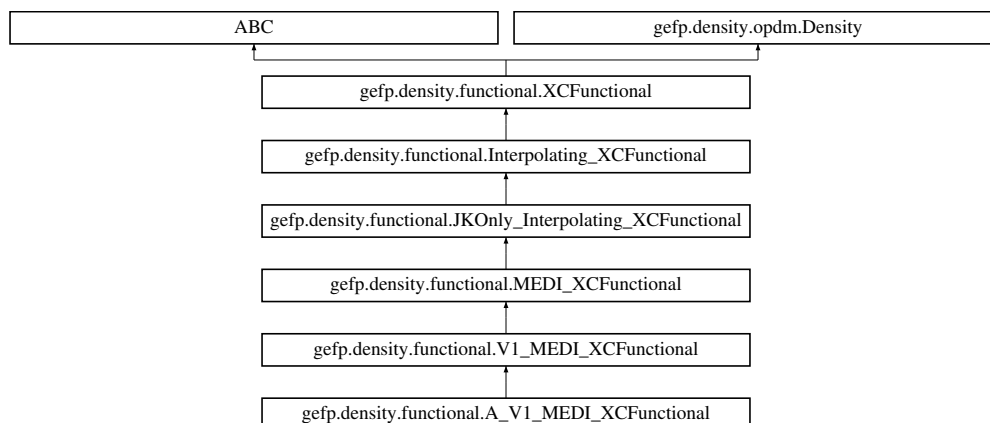
Global settings and utilities to fit DMS tensors.

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

- `gefp/gefp/density/dms.py`

## 18.3 gefp.density.functional.A\_V1\_MEDI\_XCFunctional Class Reference

Inheritance diagram for gefp.density.functional.A\_V1\_MEDI\_XCFunctional:



## Public Member Functions

- def **\_\_init\_\_** (self, coeff, kmax)
- def **abbr** (self)
- def **compute\_a0** (self, n)

## Static Public Member Functions

- def **name** ()

## Additional Inherited Members

### 18.3.1 Detailed Description

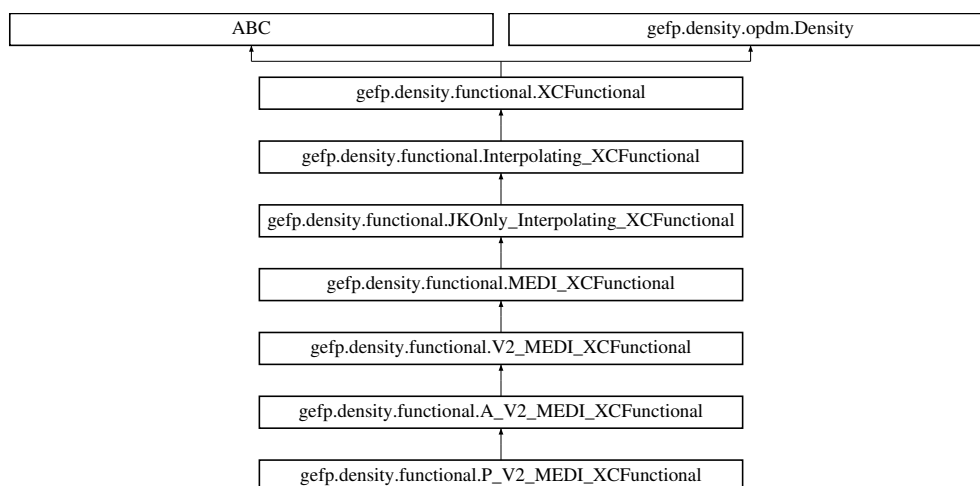
The New Class of Exchange-Correlation Functionals:  
Interpolation Functionals with Monotonous Exponential Decay.

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

- gefp/gefp/density/functional.py

## 18.4 gefp.density.functional.A\_V2\_MEDI\_XCFunctional Class Reference

Inheritance diagram for gefp.density.functional.A\_V2\_MEDI\_XCFunctional:



## Public Member Functions

- def **\_\_init\_\_** (self, coeff, kmax)
- def **abbr** (self)

- def **compute\_a0** (self, n)
- def **energy\_P\_costly** (self, x)
- def **gradient\_P** (self, x)
- def **gradient\_P\_approximate** (self, x)

## Static Public Member Functions

- def **name** ()

## Additional Inherited Members

### 18.4.1 Detailed Description

The New Class of Exchange-Correlation Functionals:  
Interpolation Functionals with Monotonous Exponential Decay.

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

- `gefp/gefp/density/functional.py`

## 18.5 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**

### 18.5.1 Detailed Description

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

- `oepdev/libutil/unitary_optimizer.h`

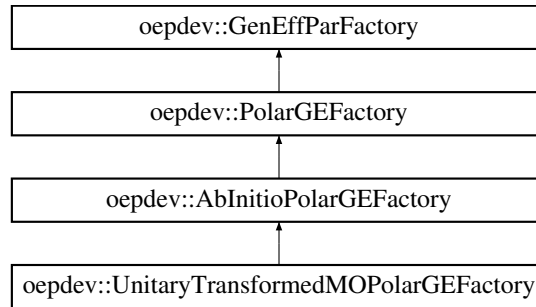


## 18.6 oepdev::AbInitioPolarGEFactory Class Reference

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

#### 18.6.1 Detailed Description

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

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

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

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

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

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

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

## 18.7 gefp.density.dms.Aggregate Class Reference

### Public Member Functions

- def **\_\_init\_\_** (self, psi4\_molecule, qm\_frgs=None)
- def **update** (self, xyz)
- def **tofile** (self, out, center\_mode=None, format='xyz', misc=None)

### Public Attributes

- **all**
- **qm\_frgs**
- **nfrags**
- **qm**
- **bath**

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

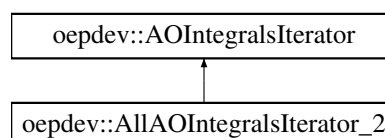
- gefp/gefp/density/dms.py

## 18.8 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

### 18.8.1 Detailed Description

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

See also

[AllAOShellCombinationsIterator\\_2](#)

### 18.8.2 Constructor & Destructor Documentation

#### 18.8.2.1 AllAOIntegralsIterator\_2() [1/2]

```
AllAOIntegralsIterator_2::AllAOIntegralsIterator_2 (
    const ShellCombinationsIterator * shellIter )
```

#### Parameters

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

### 18.8.2.2 AllAOIntegralsIterator\_2() [2/2]

```
AllAOIntegralsIterator_2::AllAOIntegralsIterator_2 (
    std::shared_ptr< ShellCombinationsIterator > shellIter )
```

#### Parameters

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

## 18.8.3 Member Function Documentation

### 18.8.3.1 index()

```
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:

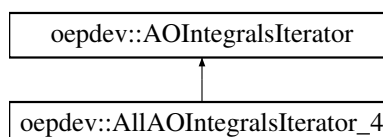
- oepdev/libutil/[integrals\\_iter.h](#)
- oepdev/libutil/integrals\_iter.cc

## 18.9 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

### 18.9.1 Detailed Description

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

See also

[AllAOShellCombinationsIterator\\_4](#)

### 18.9.2 Constructor & Destructor Documentation

#### 18.9.2.1 AllAOIntegralsIterator\_4() [1/2]

```
AllAOIntegralsIterator_4::AllAOIntegralsIterator_4 (
    const ShellCombinationsIterator * shellIter )
```

#### Parameters

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

### 18.9.2.2 AllAOIntegralsIterator\_4() [2/2]

```
AllAOIntegralsIterator_4::AllAOIntegralsIterator_4 (
    std::shared_ptr< ShellCombinationsIterator > shellIter )
```

#### Parameters

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

## 18.9.3 Member Function Documentation

### 18.9.3.1 index()

```
int oepdev::AllAOIntegralsIterator_4::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:

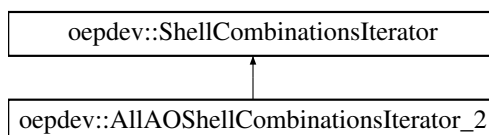
- oepdev/libutil/[integrals\\_iter.h](#)
- oepdev/libutil/integrals\_iter.cc

## 18.10 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

### 18.10.1 Detailed Description

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

### 18.10.2 Constructor & Destructor Documentation

#### 18.10.2.1 AllAOShellCombinationsIterator\_2() [1/5]

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

#### Parameters

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

**18.10.2.2 AllAOShellCombinationsIterator\_2()** [2/5]

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

**Parameters**

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

**18.10.2.3 AllAOShellCombinationsIterator\_2()** [3/5]

```
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.

**18.10.2.4 AllAOShellCombinationsIterator\_2()** [4/5]

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

**Parameters**

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

**18.10.2.5 AllAOShellCombinationsIterator\_2()** [5/5]

```
AllAOShellCombinationsIterator_2::AllAOShellCombinationsIterator_2 (
    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.

**18.10.3 Member Function Documentation****18.10.3.1 compute\_shell()**

```
void AllAOShellCombinationsIterator_2::compute_shell (
    std::shared_ptr< oepdev::TwoBodyAOInt > tei ) const [virtual]
```



## Parameters

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

Implements [oepdev::ShellCombinationsIterator](#).

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

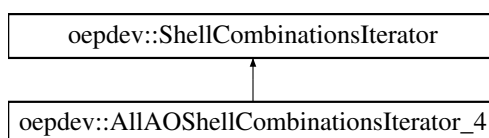
- oepdev/libutil/integrals\_iter.h
- oepdev/libutil/integrals\_iter.cc

## 18.11 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

### 18.11.1 Detailed Description

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

### 18.11.2 Constructor & Destructor Documentation

#### 18.11.2.1 AllAOShellCombinationsIterator\_4() [1/5]

```
AllAOShellCombinationsIterator_4::AllAOShellCombinationsIterator_4 (
    SharedBasisSet bs_1,
    SharedBasisSet bs_2,
    SharedBasisSet bs_3,
    SharedBasisSet bs_4 )
```

##### 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

#### 18.11.2.2 AllAOShellCombinationsIterator\_4() [2/5]

```
oepdev::AllAOShellCombinationsIterator_4::AllAOShellCombinationsIterator_4 (
    std::shared_ptr< IntegralFactory > integrals )
```

##### Parameters

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

**18.11.2.3 AllAOShellCombinationsIterator\_4()** [3/5]

```
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.

**18.11.2.4 AllAOShellCombinationsIterator\_4()** [4/5]

```
AllAOShellCombinationsIterator_4::AllAOShellCombinationsIterator_4 (
    std::shared_ptr< psi::IntegralFactory > integrals )
```

**Parameters**

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

**18.11.2.5 AllAOShellCombinationsIterator\_4()** [5/5]

```
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.

**18.11.3 Member Function Documentation****18.11.3.1 compute\_shell()** [1/2]

```
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](#).

### 18.11.3.2 compute\_shell() [2/2]

```
void oepdev::AllAOShellCombinationsIterator_4::compute_shell (
    std::shared_ptr< psi ::TwoBodyAOLnt > tei ) const [virtual]
```

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

#### Parameters

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

Implements [oepdev::ShellCombinationsIterator](#).

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

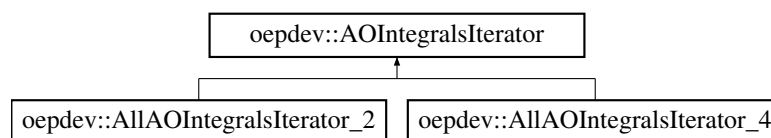
- [oepdev/libutil/integrals\\_iter.h](#)
- [oepdev/libutil/integrals\\_iter.cc](#)

## 18.12 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.*

### 18.12.1 Detailed Description

### 18.12.2 Member Function Documentation

#### 18.12.2.1 [build\(\)](#) [1/2]

```
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

### 18.12.2.2 build() [2/2]

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

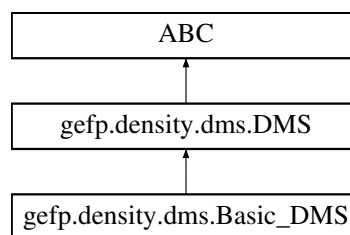
This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

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

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

## 18.13 gefp.density.dms.Basic\_DMS Class Reference

Inheritance diagram for gefp.density.dms.Basic\_DMS:



### Public Member Functions

- `def __init__ (self, type='da')`

### 18.13.1 Detailed Description

Basic model of DMS that handles induction up to second-order and first-order pure Pauli effects.

The order of DMS blocks:

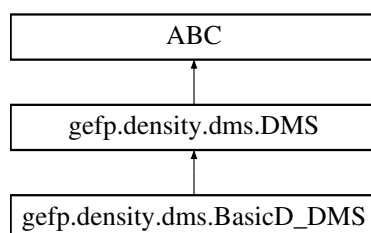
Block	DMS order	Interaction	Symmetry	Dimension	Group
-----	-----	-----	-----	-----	-----
1.	(1,0)	Induction	Symmetric	(n,n,N,3)	Z(1)
2.	(2,0)	Induction	Symmetric	(n,n,N,3,3)	Z(1)
3.	(0,1)	Pauli	Asymmetric	(n,n)	Z(2)

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

- gefp/gefp/density/dms.py

## 18.14 gefp.density.dms.BasicD\_DMS Class Reference

Inheritance diagram for gefp.density.dms.BasicD\_DMS:



### Public Member Functions

- `def __init__ (self, type='da')`

### 18.14.1 Detailed Description

Basic model of DMS that handles induction up to second-order and Pauli effects up to second-order.

The order of DMS blocks:

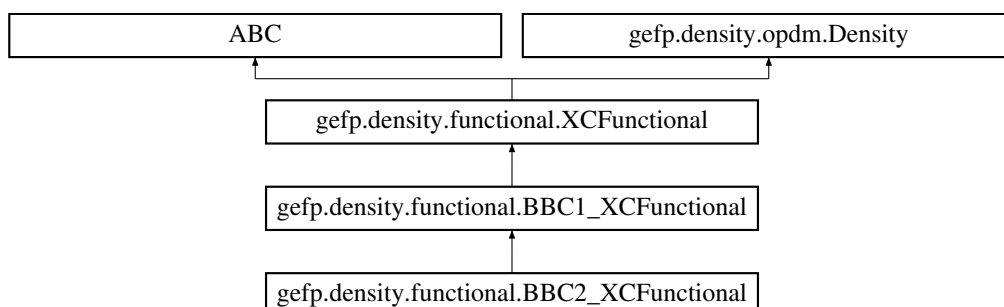
Block	DMS order	Interaction	Symmetry	Dimension	Group
-----	-----	-----	-----	-----	-----
1.	(1,0)	Induction	Symmetric	(n,n,N,3)	Z(1)
2.	(2,0)	Induction	Symmetric	(n,n,N,3,3)	Z(1)
3.	(0,2)	Pauli	Asymmetric	(n,n)	Z(1)
4.	(0,1)	Pauli	Asymmetric	(n,n)	Z(2)

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

- `gefp/gefp/density/dms.py`

## 18.15 gefp.density.functional.BBC1\_XCFunctional Class Reference

Inheritance diagram for gefp.density.functional.BBC1\_XCFunctional:



## Public Member Functions

- def `__init__` (self)
- def `abbr` (self)

## Static Public Member Functions

- def `name` ()
- def `fij` (n)

## Additional Inherited Members

### 18.15.1 Detailed Description

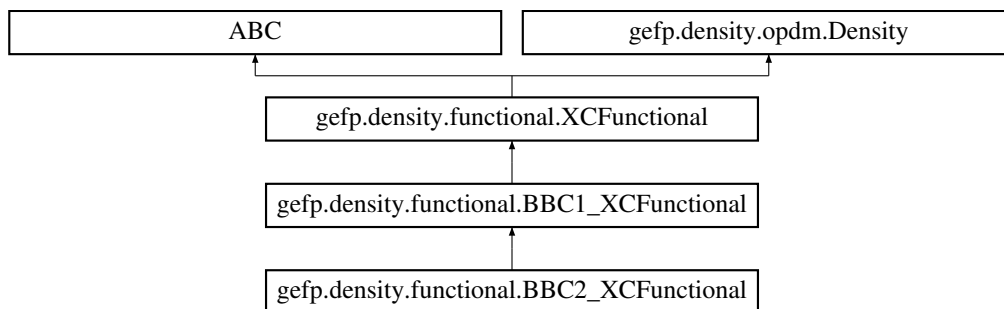
The BBC1 Exchange-Correlation Functional.

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

- `gefp/gefp/density/functional.py`

## 18.16 `gefp.density.functional.BBC2_XCFunctional` Class Reference

Inheritance diagram for `gefp.density.functional.BBC2_XCFunctional`:



## Public Member Functions

- def `__init__` (self)
- def `abbr` (self)

## Static Public Member Functions

- def `name` ()
- def `fij` (n)



## Additional Inherited Members

### 18.16.1 Detailed Description

The BBC2 Exchange-Correlation Functional.

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

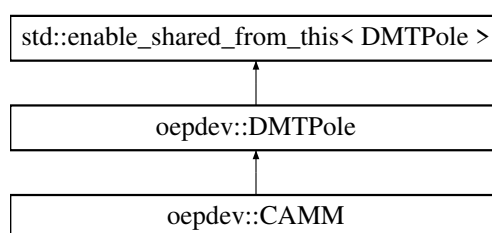
- `gefp/gefp/density/functional.py`

## 18.17 oepdev::CAMM Class Reference

Cumulative Atomic Multipole Moments.

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

Inheritance diagram for oepdev::CAMM:



## Public Member Functions

- **CAMM** (psi::SharedWavefunction wfn, int n)
- **CAMM** (const CAMM \*other)
- virtual void `compute` (psi::SharedMatrix D, bool transition, int n)  
*Compute DMTP's from the one-particle density matrix.*
- virtual void `print_header` (void) const  
*Print the header.*

## Additional Inherited Members

### 18.17.1 Detailed Description

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**

## Methodology.

The distributed multipole moments are computed in the following way:

- first the atomic additive multipole moments (AAMM's) with origins set to the global coordinate system origin are computed. AO basis set partitioning is used to distribute the AAMM's onto the atomic centres.
- subsequently, the AAMM's origins are moved to the corresponding atomic site.

The computation of the AAMM's is performed according to the following prescription:

$$M_{uw\dots z}^{(A)}(\mathbf{0}) = \sum_{\alpha \in A} \sum_{\beta \in \text{allAO's}} D_{\alpha\beta}^{\text{OED}} \langle \alpha | \mathcal{M}_{uw\dots z}(\mathbf{0}) | \beta \rangle$$

where  $M_{uw\dots z}^{(A)}$  denotes the  $(uw\dots z)$ -th component of the multipole centered at atomic site  $A$ , the symbol  $\mathcal{M}(\mathbf{0})$  is the associated quantum mechanical operator and  $D_{\alpha\beta}^{\text{OED}}$  is the (generalized) one-particle density matrix element in AO basis (Greek indices).

Recentering of the multipole moments is described in the documentation of [oepdev::DMTPole::recenter](#).

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

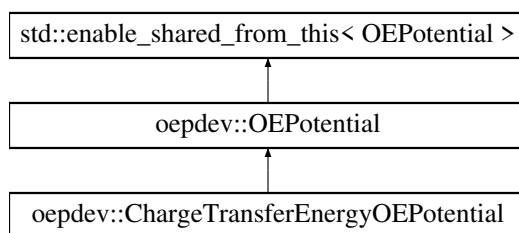
- [oepdev/lib3d/dmtp.h](#)
- [oepdev/lib3d/dmtp\\_camm.cc](#)

## 18.18 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

*Header information.*

## Additional Inherited Members

### 18.18.1 Detailed Description

Contains the following OEP types:

- `Otto-Ladik.V1.GDF` - DF-based term (group I)
- `Otto-Ladik.V3.CAMM-nj` - CAMM-based term (group III; truncated on distributed charges)

Group II terms do not require any particular OEP's due to great simplification of this term. Atomic numbers and LMO centroids are sufficient.

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

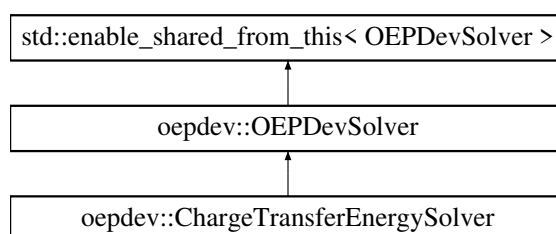
- `oepdev/liboep/oep.h`
- `oepdev/liboep/oep_energy_ct.cc`

## 18.19 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

### 18.19.1 Detailed Description

The implemented methods are shown below

Table 18.15: Methods available in the Solver

Keyword	Method Description
<b>Benchmark Methods</b>	
OTTO_LADIK	<i>Default.</i> 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	<i>Default.</i> OEP-based Otto-Ladik expressions.

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

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

where

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

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

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

where

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

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).

## 18.19.2 Member Function Documentation

### 18.19.2.1 compute\_benchmark()

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

Each solver object has one `DEFAULT` benchmark method

#### Parameters

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

Implements [oepdev::OEPDevSolver](#).

### 18.19.2.2 compute\_oep\_based()

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

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

#### Parameters

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

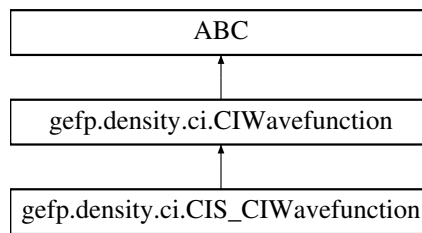
Implements [oepdev::OEPDevSolver](#).

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

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

## 18.20 gefp.density.ci.CIS\_CIWavefunction Class Reference

Inheritance diagram for gefp.density.ci.CIS\_CIWavefunction:



### Public Member Functions

- `def __init__ (self, ref_wfn, E, W)`
- `def make_ci_l (self)`

### Additional Inherited Members

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

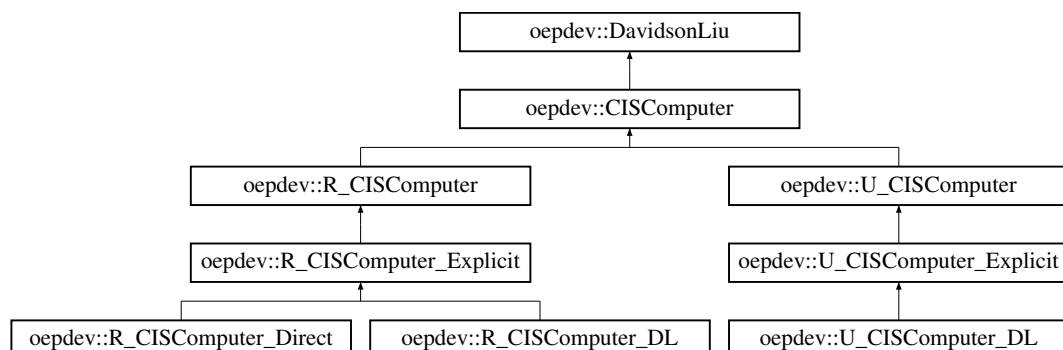
- `gefp/gefp/density/ci.py`

## 18.21 oepdev::CISComputer Class Reference

[CISComputer](#).

```
#include <cis.h>
```

Inheritance diagram for oepdev::CISComputer:



### Public Member Functions

- `virtual ~CISComputer ()`

*Destructor.*

- virtual void `compute` (void)

*Solve the CIS problem.*

- virtual void `clear_dpd` (void)

*Clear DPD instance.*

- int `nstates` (void) const

*Get the total number of excited states.*

- psi::SharedVector `eigenvalues` () const

*Get the CIS eigenvalues.*

- psi::SharedVector `E` () const

- psi::SharedMatrix `eigenvectors` () const

*Get the CIS eigenvectors.*

- psi::SharedMatrix `U` () const

- std::pair< double, double > `U_homo_lumo` (int l, int h=0, int l=0) const

*Get the HOMO+\*h\*->LUMO+\*l\* CIS coefficient for a given excited state l for spin alpha and beta.*

- SharedMatrix `Da_mo` (int i) const

*Compute MO one-particle alpha density matrix for state i*

- SharedMatrix `Db_mo` (int i) const

*Compute MO one-particle beta density matrix for state i*

- SharedMatrix `Da_ao` (int i) const

*Compute AO one-particle alpha density matrix for state i*

- SharedMatrix `Db_ao` (int i) const

*Compute AO one-particle beta density matrix for state i*

- SharedDMTPole `cammm` (int j, bool symmetrize=false) const

*Compute CAMM for j excited state.*

- SharedMatrix `Ta_ao` (int j) const

*Compute MO one-particle alpha 0->\*j\* transition density matrix.*

- SharedMatrix `Tb_ao` (int j) const

*Compute MO one-particle beta 0->\*j\* transition density matrix.*

- SharedMatrix `Ta_ao` (int i, int j) const

*Compute MO one-particle alpha i->\*j\* transition density matrix.*

- SharedMatrix `Tb_ao` (int i, int j) const

*Compute MO one-particle beta i->\*j\* transition density matrix.*

- SharedDMTPole `trcammm` (int j, bool symmetrize=true) const

*Compute TrCAMM for 0->\*j\* transition.*

- SharedDMTPole `trcammm` (int i, int j, bool symmetrize=true) const

*Compute TrCAMM for i->\*j\* transition.*

- SharedVector `transition_dipole` (int j) const

*Compute transition dipole moment for 0->\*j\* transition.*

- SharedVector `transition_dipole` (int i, int j) const



- Compute transition dipole moment for  $i \rightarrow j$  transition.*
- double [oscillator\\_strength](#) (int j) const  
*Compute oscillator strength for  $0 \rightarrow j$  transition.*
- double [oscillator\\_strength](#) (int i, int j) const  
*Compute oscillator strength for  $i \rightarrow j$  transition.*
- double [s2](#) (int i) const  
*Compute  $\langle S^2 \rangle$  expectation value for the  $i$ th state.*
- void [determine\\_electronic\\_state](#) (int &l)  
*Determine electronic state.*
- std::shared\_ptr< [CISData](#) > [data](#) (int l, int h, int l, bool symmetrize\_trcamm=false)  
*Return CIS data structure for a given excited state l*

## Static Public Member Functions

- static std::shared\_ptr< [CISComputer](#) > [build](#) (const std::string &type, std::shared\_ptr< psi::Wavefunction > wfn, psi::Options &opt, const std::string &reference="")  
*Build CIS Computer.*

## Static Public Attributes

- static const std::vector< std::string > [reference\\_types](#) = {"RHF", "UHF"}  
*Slater determinant possible references, that are implemented.*

## Protected Member Functions

- **CISComputer** (std::shared\_ptr< psi::Wavefunction > wfn, psi::Options &opt, psi::IntegralTransform::Transform\_type trans\_type)
- virtual void **print\_header\_** (void)
- virtual void **set\_nstates\_** (void)
- virtual void **allocate\_memory** (void)
- virtual void **allocate\_hamiltonian\_** (void)
- virtual void **prepare\_for\_cis\_** (void)
- virtual void **build\_hamiltonian\_** (void)=0
- virtual void **diagonalize\_hamiltonian\_** (void)
- virtual void **standardize\_amplitudes\_** (void)
- virtual void **print\_excited\_states\_** (void)
- virtual void **print\_excited\_state\_character\_** (int l)=0
- virtual void **set\_beta\_** (void)=0
- virtual void **transform\_integrals\_** (void)
- virtual void **davidson\_liu\_compute\_diagonal\_hamiltonian** (void)
- virtual void **davidson\_liu\_compute\_sigma** (void)

## Protected Attributes

- `std::shared_ptr< psi::Wavefunction > ref_wfn_`  
*Reference wavefunction.*
- `const int nmo_`  
*Psi4 Options.*
- `const int naocc_`  
*Number of alpha occupied MO's.*
- `const int nbocc_`  
*Number of beta occupied MO's.*
- `const int navir_`  
*Number of alpha virtual MO's.*
- `const int nbvir_`  
*Number of beta virtual MO's.*
- `int ndets_`  
*Number of excited determinants.*
- `int nstates_`  
*Number of excited states.*
- `SharedMatrix H_`  
*CIS Excited State Hamiltonian in Slater determinantal basis.*
- `SharedMatrix U_`
- `SharedVector E_`
- `std::shared_ptr< psi::JK > jk_`  
*Computer of generalized JK objects.*
- `SharedVector eps_a_o_`
- `SharedVector eps_a_v_`
- `SharedVector eps_b_o_`
- `SharedVector eps_b_v_`
- `const psi::IntegralTransform::TransformationType transformation_type_`  
*MO Integral Transformation Type.*
- `std::shared_ptr< psi::IntegralTransform > inttrans_`

### 18.21.1 Detailed Description

### 18.21.2 Member Function Documentation

## 18.21.2.1 build()

```
std::shared_ptr< CISComputer > oepdev::CISComputer::build (
    const std::string & type,
    std::shared_ptr< psi::Wavefunction > wfn,
    psi::Options & opt,
    const std::string & reference = "" ) [static]
```

## Parameters

<i>type</i>	- Type of computer
<i>wfn</i>	- Psi4 wavefunction
<i>opt</i>	- Psi4 options
<i>reference</i>	- Reference Slater determinant (RHF, UHF available).

Available computer types:

- RESTRICTED or RCIS - RHF wavefunction is used as reference state
- UNRESTRICTED or UCIS - UHF wavefunction is used as reference state

## Implementation

The CIS Hamiltonian in the basis space of singly-excited Slater determinants is constructed from canonical molecular orbitals (CMO's)

$$\begin{aligned}\langle \Phi_0 | \mathcal{H} | \Phi_i^a \rangle &= 0 \\ \langle \Phi_j^b | \mathcal{H} | \Phi_i^a \rangle &= \delta_{ij} \delta_{ab} (\epsilon_a - \epsilon_i) + \langle aj | ib \rangle - \langle aj | bi \rangle\end{aligned}$$

where  $i$  labels the occupied CMO's whereas  $a$  labels the virtual CMO's. In the above equation,  $\langle aj | ib \rangle$  is the 2-electron 4-centre integral in physicist's notation. After integrating out the spin coordinate, four blocks of Hamiltonian are explicitly given as

$$\begin{aligned}\langle \Phi_j^b | \mathcal{H} | \Phi_i^a \rangle &= \delta_{ij} \delta_{ab} (\epsilon_a - \epsilon_i) + [ia | jb] - [ab | ij] \\ \langle \Phi_{\bar{j}}^{\bar{b}} | \mathcal{H} | \Phi_i^{\bar{a}} \rangle &= \delta_{\bar{i}\bar{j}} \delta_{\bar{a}\bar{b}} (\epsilon_a - \epsilon_i) + [\bar{i}\bar{a} | \bar{j}\bar{b}] - [\bar{a}\bar{b} | \bar{i}\bar{j}] \\ \langle \Phi_{\bar{j}}^{\bar{b}} | \mathcal{H} | \Phi_i^a \rangle &= [ia | \bar{j}\bar{b}] \\ \langle \Phi_j^b | \mathcal{H} | \Phi_i^{\bar{a}} \rangle &= [\bar{i}\bar{a} | jb]\end{aligned}$$

where the  $[ia | jb]$  is the 2-electron 4-centre integral in the chemist's (Coulomb) notation.

Such matrix is diagonalized yielding the excitation energies (wrt HF ground state) as well as the CIS coefficients

$$\sum_{ij} \sum_{ab} t_{ij}^a H_{ij}^{ab} t_{j,J}^b = E_I \delta_{IJ}$$

where the summations above extend over alpha and beta electron spin labels and  $t_{i,I}^a$  is the CIS amplitude for the  $i \rightarrow a$  excited state, associated with the  $i \rightarrow a$  excitation with respect to the HF reference determinant. Note that  $E_I$  is *not* the excited state energy, but the energy relative the the HF reference energy.

See also

For Davidson-Liu solution to CIS problem, see [oepdev::R\\_CISComputer\\_DL](#) and [oepdev::U\\_CISComputer\\_DL](#).

### Transition density matrix

AO basis transition density from ground (HF) to excited (CIS) state is given by

$$P_{\mu\nu}^{(g \rightarrow e)} = \sum_i^{\text{Occ}} \sum_a^{\text{Vir}} t_{i,e}^a C_{\nu i} C_{\mu a} + \sum_{\bar{i}}^{\text{Occ}} \sum_{\bar{a}}^{\text{Vir}} t_{\bar{i},e}^{\bar{a}} C_{\nu \bar{i}} C_{\mu \bar{a}}$$

### Excited state density matrix

CMO basis excited state density matrix for alpha spin is given by

Analogous expression is given for the beta spin.

AO representation of the CMO excited state density matrix is

$$P_{\mu\nu}^{(e)} = \sum_{pq} C_{\mu p} P_{pq}^{(e)} C_{\nu q} + \sum_{\bar{p}\bar{q}} C_{\mu \bar{p}} P_{\bar{p}\bar{q}}^{(e)} C_{\nu \bar{q}}$$

which is the sum of alpha and beta density matrices in CMO basis transformed to AO basis.

The CMO excited state density matrix for spin alpha is given by

$$P_{pq}^{(e)} = \begin{cases} \delta_{pq} - \sum_a^{\text{Vir}} t_{p,e}^a t_{q,e}^a & \text{for } p, q \in \text{Occ} \\ \sum_i^{\text{Occ}} t_{i,e}^p t_{i,e}^q & \text{for } p, q \in \text{Vir} \\ 0 & \text{otherwise} \end{cases}$$

The beta spin density matrix is generated analogously as above.

The cumulative atomic multipole moments ([CAMM](#)) are computed from the excited state density matrices in AO basis. The nuclear contribution is included.

### Transition multipole moments

The transition dipole moment is computed from the AO transition density matrix and the dipole integrals in AO basis, i.e.,

$$\langle \Phi_0 | \hat{\mu}_u | \Psi_e \rangle = \text{Tr} \left[ \mathbf{d}^{(u)} \cdot \mathbf{P}^{g \rightarrow e} \right]$$

Oscillator strength is computed from the transition dipole moment via

$$f^{g \rightarrow e} = \frac{2}{3} E_e \left| \langle \Phi_0 | \hat{\mu} | \Psi_e \rangle \right|^2$$

Transition cumulative atomic multipole moments (TrCAMM) are computed from the transition density matrices in AO basis. The nuclear contribution is not included.

## Spin angular momentum

The expectation value of the  $\hat{S}^2$  operator is calculated from the CIS amplitudes and MOs of the reference wavefunction according to D. Maurice and M. Head-Gordon, *Int. J. Quant. Chem.*, **1995**, 95, 010361-10:

$$\begin{aligned} \langle \hat{S}^2 \rangle_{\text{UCIS}} = \langle \hat{S}^2 \rangle_{\text{UHF}} - \text{Tr} \left[ \mathbf{Q}_{\text{Occ}}^{(\alpha)} \cdot \left\{ \mathbf{P}_{\text{Occ}}^{(e,\alpha)} - \mathbf{1} \right\} \right] - \text{Tr} \left[ \mathbf{Q}_{\text{Occ}}^{(\beta)} \cdot \left\{ \mathbf{P}_{\text{Occ}}^{(e,\beta)} - \mathbf{1} \right\} \right] \\ - \text{Tr} \left[ \mathbf{Q}_{\text{Vir}}^{(\alpha)} \cdot \mathbf{P}_{\text{Vir}}^{(e,\alpha)} \right] - \text{Tr} \left[ \mathbf{Q}_{\text{Vir}}^{(\beta)} \cdot \mathbf{P}_{\text{Vir}}^{(e,\beta)} \right] - 2 \sum_i^{\text{Occ}} \sum_a^{\text{Vir}} \sum_{\bar{j}}^{\text{Occ}} \sum_{\bar{b}}^{\text{Vir}} \Delta_{i\bar{j}}^* \Delta_{a\bar{b}} t_{i,e}^a t_{\bar{j},e}^{\bar{b}} \end{aligned}$$

where

$$\begin{aligned} [\mathbf{Q}_{\text{Occ}}^{(\alpha)}]_{ij} &= \sum_{\bar{k}}^{\text{Occ}} \Delta_{\bar{k}i}^* \Delta_{\bar{k}j} \\ [\mathbf{Q}_{\text{Occ}}^{(\beta)}]_{i\bar{j}} &= \sum_k^{\text{Occ}} \Delta_{ki}^* \Delta_{k\bar{j}} \\ [\mathbf{Q}_{\text{Vir}}^{(\alpha)}]_{ab} &= \sum_{\bar{k}}^{\text{Occ}} \Delta_{\bar{k}a}^* \Delta_{\bar{k}b} \\ [\mathbf{Q}_{\text{Vir}}^{(\beta)}]_{\bar{a}\bar{b}} &= \sum_k^{\text{Occ}} \Delta_{k\bar{a}}^* \Delta_{k\bar{b}} \end{aligned}$$

and

$$\Delta_{pq} = \sum_{\mu\nu} C_{\mu p} S_{\mu\nu} C_{\nu q}$$

The diagnostic for UHF spin contamination is given by

$$\langle \hat{S}^2 \rangle_{\text{UHF}} = \langle \hat{S}^2 \rangle_{\text{exact}} + N_\beta - \sum_i^{\text{Occ}} \sum_{\bar{j}}^{\text{Occ}} |\Delta_{i\bar{j}}|^2$$

with

$$\langle \hat{S}^2 \rangle_{\text{exact}} = \frac{N_\alpha - N_\beta}{2} \left( \frac{N_\alpha - N_\beta + 2}{2} \right)$$

and is also printed out to the output file.

## Note

Useful options:

- `CIS_TYPE` - Algorithm of CIS. Available: `DAVIDSON_LIU` (Default), `DIRECT_EXPLICIT` (only RHF reference), `EXPLICIT`.
- `CIS_SCHWARTZ_CUTOFF` - Cutoff for Schwartz ERI screening. Default: 0.0. Relevant if `DAVIDSON_LIU` or `DIRECT_EXPLICIT` are chosen as CIS type.
- `CIS_STANDARDIZE_AMPLITUDES` - If true, CIS amplitudes of each excited state are rephased so that the leading amplitude is positive. Default: true.
- `OEPDEV_AMPLITUDE_PRINT_THRESHOLD` - Control threshold how many CIS amplitudes to print to the output. Default: 0.1.
- For UHF references, SAD guess might lead to triplet instabilities. It is then better to set `CORE` as the UHF guess

## 18.21.3 Member Data Documentation

### 18.21.3.1 nmo\_

```
const int oepdev::CISComputer::nmo_ [protected]
```

Number of MO's

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

- oepdev/libutil/[cis.h](#)
- oepdev/libutil/cis\_base.cc

## 18.22 oepdev::CISData Struct Reference

Container to handle the CIS wavefunction parameters.

```
#include <cis.h>
```

### Public Attributes

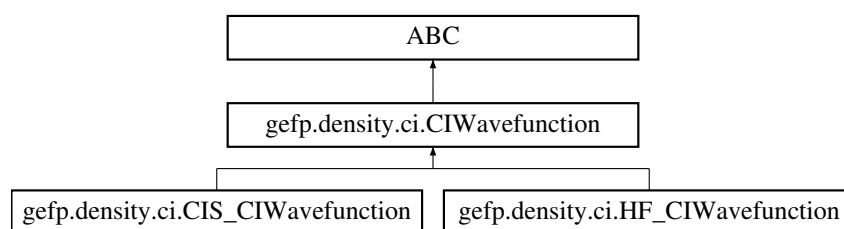
- double [E\\_ex](#)  
*Excitation energy.*
- double [t\\_homo\\_lumo](#)  
*CIS HOMO-LUMO amplitude.*
- SharedMatrix [Pe](#)  
*Excited state density matrix (sum of alpha and beta)*
- SharedMatrix [Peg](#)  
*Transition ground-to-excited state density matrix (sum of alpha and beta)*
- SharedDMTPole [trcamm](#)  
*TrCAMM.*
- SharedDMTPole [camm\\_homo](#)  
*CAMM for HOMO orbital.*
- SharedDMTPole [camm\\_lumo](#)  
*CAMM for LUMO orbital.*

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

- oepdev/libutil/[cis.h](#)

## 18.23 gefp.density.ci.CIWavefunction Class Reference

Inheritance diagram for gefp.density.ci.CIWavefunction:



### Public Member Functions

- def **\_\_init\_\_** (self, ref\_wfn, E, W)
- def **make\_ci\_l** (self)
- def **overlap** (self, other)

### Public Attributes

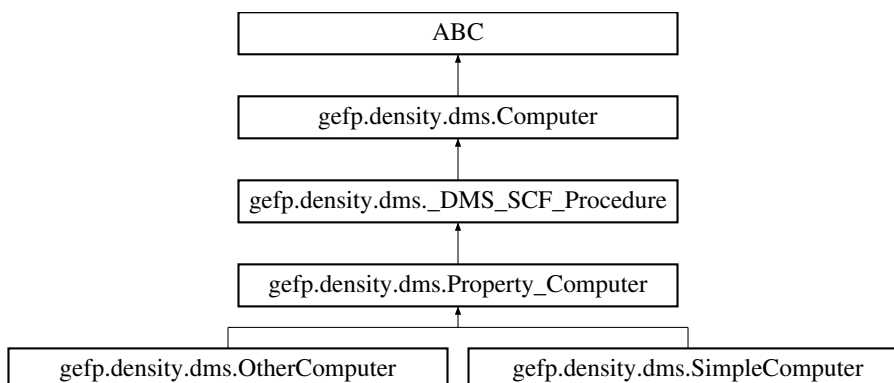
- **ref\_wfn**
- **ci\_e**
- **ci\_c**
- **ca\_o**
- **cb\_o**
- **ca\_v**
- **cb\_v**
- **bfs**
- **naocc**
- **nbocc**
- **nmo**
- **navir**
- **nbvir**
- **ci\_l**
- **ndet**

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

- gefp/gefp/density/ci.py

## 18.24 gefp.density.dms.Computer Class Reference

Inheritance diagram for gefp.density.dms.Computer:



### Public Member Functions

- `def __init__ (self, aggregate, fragments)`
- `def run (self, field=None, verbose=None, converge_by='energy')`
- `def update (self, psi_molecule)`
- `def Da (self)`
- `def Da.0 (self)`
- `def Fa (self)`
- `def total_energy (self)`
- `def E (self)`

### Static Public Attributes

- `bool verbose = True`
- `float e_convergence = 1.0e-6`
- `int dmsscf_maximum_number_of_iterations = 1000`
- `bool raise_error_when_unconverged = True`

#### 18.24.1 Detailed Description

---

Method to solve DMS-SCF problems.

Usage:

```

computer = Computer(aggregate, fragments)
energy = computer.run(field=[0,0,0])
mu      = computer.dipole_moment()
mu      = computer.ff_dipole_moment()
alpha   = computer.ff_polarizability(energy=False)
beta    = computer.ff_hyperpolarizability(energy=False)
  
```



```

gamma = computer.ff_hyper2polarizability(energy=Fanse)
camm = computer.camm()
charges= computer.atomic_charges(kappa=0.0)

computer.update(psi_molecule)

```

---

Gundelfingen, 31

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

- `gefp/gefp/density/dms.py`

## 18.25 oepdev::CPHF Class Reference

**CPHF** solver class.

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

### Public Member Functions

#### Constructor and Destructor

- **CPHF** (SharedWavefunction ref\_wfn, Options &options)  
*Constructor.*
- $\sim$ **CPHF** ()  
*Destructor.*

#### Executor

- void **compute** (void)  
*run the calculations*

#### Printer

- void **print** (void) const  
*print to output file*

#### Accessors

- 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 (always Canonical)*
- `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

### Basic Data

- `std::shared_ptr< psi::Wavefunction > _wfn`  
*Wavefunction object.*
- Options & `_options`  
*Options.*
- `std::shared_ptr< BasisSet > _primary`  
*Primary Basis Set.*
- `std::shared_ptr< Localizer > _localizer`  
*Orbital localizer.*

### Sizing Information

- `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.*

### Parameters of CPHF Calculations

- 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.*

### Molecular Orbitals

- 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::shared\_ptr< psi::Matrix > [\\_T](#)
- *Transformation from old to new MO's.*

### DIIS Manager

- std::vector< std::shared\_ptr< [oepdev::DIISManager](#) > > [\\_diis](#)
- *the DIIS managers for each perturbation operator x, y and z*

### Response Properties

- 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.*

### 18.25.1 Detailed Description

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` - whether 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`

### 18.25.2 Constructor & Destructor Documentation

#### 18.25.2.1 CPHF()

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

#### 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:

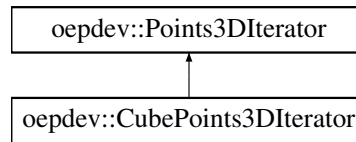
- `oepdev/libutil/cphf.h`
- `oepdev/libutil/cphf.cc`

## 18.26 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

### 18.26.1 Detailed Description

**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:

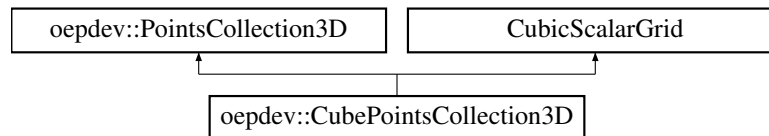
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

## 18.27 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

#### 18.27.1 Detailed Description

**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:

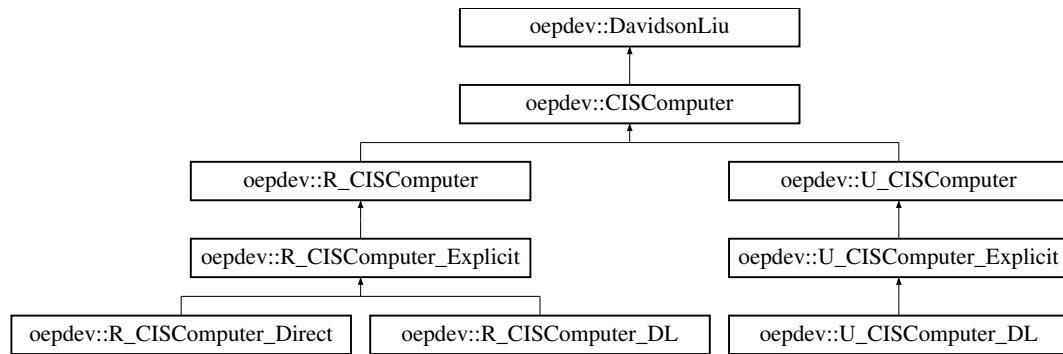
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

## 18.28 oepdev::DavidsonLiu Class Reference

Davidson-Liu diagonalization method.

```
#include <davidson_liu.h>
```

Inheritance diagram for oepdev::DavidsonLiu:



## Public Member Functions

- [DavidsonLiu](#) (psi::Options &opt)  
*Constructor.*
- virtual [~DavidsonLiu](#) ()  
*Destructor.*
- virtual void [run\\_davidson\\_liu](#) ()  
*Run the Davidson-Liu solver.*
- psi::SharedVector [eigenvalues\\_davidson\\_liu](#) () const  
*Get the eigenvalues.*
- psi::SharedVector **E\_davidson\_liu** () const
- psi::SharedMatrix [eigenvectors\\_davidson\\_liu](#) () const  
*Get the eigenvectors.*
- psi::SharedMatrix **U\_davidson\_liu** () const

## Protected Member Functions

- virtual void [davidson\\_liu\\_initialize](#) (int N, int L, int M)  
*Helper interface.*
- virtual void **davidson\_liu\_initialize\_guess\_vectors** ()
- virtual void **davidson\_liu\_initialize\_guess\_vectors\_by\_random** ()
- virtual void **davidson\_liu\_initialize\_guess\_vectors\_by\_custom** ()
- virtual void **davidson\_liu\_compute\_diagonal\_hamiltonian** ()=0
- virtual void **davidson\_liu\_compute\_sigma** ()=0
- virtual void **davidson\_liu\_add\_guess\_vectors** ()
- virtual double **davidson\_liu\_compute\_convergence** ()
- virtual void **davidson\_liu\_finalize** (bool)

## Protected Attributes

- int [N\\_davidson\\_liu\\_](#)  
*Dimensionality of Hamiltonian.*
- int [L\\_davidson\\_liu\\_](#)  
*Number of guess vectors.*
- int [M\\_davidson\\_liu\\_](#)  
*Number of roots of interest.*
- psi::Options & [options\\_](#)  
*Psi4 options.*
- psi::SharedVector [E\\_davidson\\_liu\\_](#)  
*Eigenvalues.*
- psi::SharedMatrix [U\\_davidson\\_liu\\_](#)  
*Eigenvectors.*
- psi::SharedVector [H\\_diag\\_davidson\\_liu\\_](#)  
*Diagonal elements of the matrix to diagonalize.*
- psi::SharedVector [E\\_old\\_davidson\\_liu\\_](#)  
*Old estimation of eigenvalues.*
- bool [davidson\\_liu\\_initialized\\_](#)  
*Is Davidson-Liu computer ready for calculations?*
- bool [davidson\\_liu\\_finalized\\_](#)  
*Is Davidson-Liu computer finished with calculations?*
- int **davidson.liu.n\_sigma\_computed\_**
- std::vector< psi::SharedVector > [sigma\\_vectors\\_davidson\\_liu\\_](#)  
*Sigma vectors stored.*
- std::shared\_ptr< [oepdev::GramSchmidt](#) > [guess\\_vectors\\_davidson\\_liu\\_](#)  
*Object storing guess vectors.*

### 18.28.1 Detailed Description

Find the lowest  $M$  eigenvalues and associated eigenvectors of the real, symmetric (square) matrix **H**.

Associated options:

- DAVIDSON\_LIU\_NROOTS - number of roots of interest. Default: 1.
- DAVIDSON\_LIU\_CONVER - convergence of the iterative procedure as RMS of old and current eigenvalues. Default: 1.0E-10.
- DAVIDSON\_LIU\_MAXITER - maximum number of iterations. Default: 500.
- DAVIDSON\_LIU\_GUESS - Type of guess vectors. Default: RANDOM, which is constructing random vectors.



- `DAVIDSON_LIU_THRESH_LARGE` - Small correction vector threshold (see description below). Default: 1.0E-03.
- `DAVIDSON_LIU_THRESH_SMALL` - Small correction vector threshold (see description below). Default: 1.0E-06.
- `DAVIDSON_LIU_SPACE_MAX` - Maximum number of guess vectors. Default: 200.
- `DAVIDSON_LIU_SPACE_START` - Starting amount of guess vectors. Must be larger or equal to number of roots. Default: -1, which means that number of roots is taken.
- `DAVIDSON_LIU_STOP_WHEN_UNCONVERGED` - Raise error when iterations do not converge. Default: True.

## Usage in C++ programming

This class is an abstract base. In order to use the Davidson-Liu method fully implemented here, one must define a child class inheriting from [oepdev::DavidsonLiu](#) and implementing two of the pure methods:

- `davidson_liu_compute_diagonal_hamiltonian` - method specifying the calculation of the  $\sigma$  vectors, which are stored in the `std::vector<psi::SharedVector> sigma_vectors_davidson_liu;`
- `davidson_liu_compute_diagonal_hamiltonian` - method specifying the calculation of the diagonal elements of the Hamiltonian, stored in the `psi::SharedVector H_diag_davidson_liu.`

## See also

Examples for demo use.

## Implementation

The implementation follows Figure 5, Section 3.2.1 in Ref.[1]. Dimensionality:

- $N$  - number of rows/columns of matrix to diagonalize
- $L$  - current number of guess vectors
- $M$  - number of roots of interest

Sigma vectors are defined to be

$$\mathbf{S} = \mathbf{H}\mathbf{B}$$

where  $\mathbf{B}$  are the guess vectors stored as a matrix of size  $(N, L)$  in core memory. Subspace Hamiltonian is then given by

$$\mathbf{G} = \mathbf{B}^T \mathbf{S}$$

and is diagonalized using standard diagonalization technique,

$$\mathbf{G} = \mathbf{U}\mathbf{z}\mathbf{U}^T$$

where  $\mathbf{z}$  are the eigenvalues. First  $M$  lowest eigenvalues and associated eigenvectors are saved in  $\mathbf{E}$  and  $\mathbf{A}$ , respectively (with the latter having size of  $(L, M)$ ). The current eigenvector matrix  $\mathbf{C}$  containing roots is given by

$$\mathbf{C} = \mathbf{B}\mathbf{A}$$

Once this step is completed, the correction vectors are computed for each eigenvalue according to

$$\delta_{Ik} = \frac{1}{E_k - H_{II}} \left[ -E_k C_{Ik} + \sum_l^L \sigma_{Il} A_{lk} \right]$$

and they are orthonormalized against all the columns of  $\mathbf{B}$  by using the Gram-Schmidt procedure. If the norm of such orthonormalized correction vector is larger than threshold value, it is appended to  $\mathbf{B}$  as new guess vector.

#### Note

Note that the current implementation uses the original Davidson's preconditioner, which might have problems with breaking spin symmetry of the solution.

#### Treatment of correction vector threshold.

In the current implementation, two threshold values are defined:

- larger threshold, controlled by `DAVIDSON_LIU_THRESH_LARGE` Psi4 option, is used for the first lowest eigenvalue.
- smaller threshold, controlled by `DAVIDSON_LIU_THRESH_SMALL` Psi4 option, is used for the next eigenvalues if  $M > 1$ .

## References

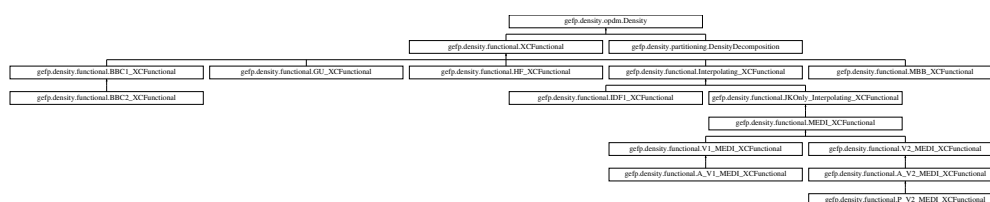
[1] C. David Sherrill and Henry F. Schaefer III, *Adv. Quant. Chem.* **1999** (34), pp. 94720-1460.

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

- `oepdev/libutil/davidson.liu.h`
- `oepdev/libutil/davidson.liu.cc`

## 18.29 gefp.density.opdm.Density Class Reference

Inheritance diagram for gefp.density.opdm.Density:



## Public Member Functions

- `def __init__ (self, D=None, jk=None)`
- `def matrix (self)`
- `def set_D (self, D)`
- `def set_jk (self, jk)`
- `def compute_1el_energy (self, D, Hcore)`
- `def compute_2el_energy (self, D_left, D_right, type='j')`
- `def generalized_JK (self, D, type='j')`
- `def natural_orbitals (cls, D, S=None, C=None, orthogonalize_mo=True, order='descending', return_ao_orthogonal=False, renormalize=False, no_cutoff=False, ignore_large_n=False, n_eps=5.0E-5)`
- `def generalized_density (cls, n, c, g=1.0)`
- `def orthogonalize_OPDM (cls, D, S)`
- `def deorthogonalizer (cls, S)`
- `def orthogonalizer (cls, S)`

### 18.29.1 Detailed Description

---

Electron Density

Handles the Electron Density Distribution.

---

Usage as container class:

1) Initialize container object:

```
density = Density(D = None, jk = None)
```

where:

- D - the density matrix in AO or MO basis
- jk - the psi4::JK object for AO basis JK calculations

2) Grab the density matrix

```
D = density.matrix()
```

3) Computations in AO basis:

o compute 1-electron energy (does not require JK object to be set)

```
e_1 = density.compute_1el_energy(D, V1)
```

The below require jk to be set:

o compute 2-electron energy (J-type expression)

```
e_2j = density.compute_2el_energy(D_left, D_right, type='j')
```

o compute 2-electron energy (K-type expression)

```
e_2k = density.compute_2el_energy(D_left, D_right, type='k')
```

o compute J matrix (or K matrix if type=='k'):

```
J = density.generalized_JK(D, type='j')
```

---

Usage as method class.

Using 'Density' as a class of methods do not require object initialization.  
The list of class methods is given below:

o Density.natural_orbitals	- compute natural orbitals
o Density.generalized_density	- compute generalized OPDM
o Density.orthogonalize_OPDM	- compute orthogonalized OPDM
o Density.orthogonalizer	- compute orthogonalizer matrix
o Density.deorthogonalizer	- compute deorthogonalizer rmatrix

Usage:

```
result = Density.'class method name'
```

See respective documentation for each of them for further details.

---

Last Revis

## 18.29.2 Member Function Documentation

### 18.29.2.1 natural\_orbitals()

```
def gefp.density.opdm.Density.natural_orbitals (
    cls,
    D,
    S = None,
    C = None,
    orthogonalize_mo = True,
    order = 'descending',
    return_ao_orthogonal = False,
```

```

    renormalize = False,
    no_cutoff = False,
    ignore_large_n = False,
    n_eps = 5.0E-5 )

```

---

Compute the Natural Orbitals from a given OPDM

---

Usage:

```

n, c = Density.natural_orbitals(D, S = None, C = None,
                                orthogonalize_mo = True,
                                order = 'descending',
                                return_ao_orthogonal = False,
                                renormalize = False,
                                no_cutoff = False,
                                ignore_large_n = False,
                                n_eps = 5.0E-5)

```

where:

- o D - OPDM in AO or MO basis
  - o S - overlap integrals in AO or MO basis
  - o C - LCAO-MO transformation matrix
  - o orthogonalize\_mo - whether to transform D from AO to certain MO basis and diagonalize
  - o order - order in which eigenvalues (occupancies) are sorted. Eigenvalues are sorted in descending order
  - o return\_ao\_orthogonal - whether to return NO's in oAO basis set or not
  - o renormalize - renormalize to integer number of electrons
  - o no\_cutoff - cut-off threshold for occupancies
  - o ignore\_large\_n - raise ValueError if  $(1.0 + n\_eps) < n < (0.0 - n\_eps)$
  - o n\_eps - tolerance for occupancy deviation
- 

Examples:

- 1) NO's in AO (non-orthogonal, original) basis from D in AO basis

```

n, c = Density.natural_orbitals(D, S, C, orthogonalize_mo = True, n_eps = 0.001)

```

D: ndarray of shape (AO x AO)

S: ndarray of shape (AO x AO)

C: ndarray of shape (AO x MO)

--> transformation D (MO x MO) = C.T S D S C and its diagonalization

--> transformation of transformation matrix from MO to AO basis

n: ndarray of shape (NO)

c: ndarray of shape (AO x NO)

- 2) NO's in certain orthogonal MO basis from D in the same MO basis

```

n, c = Density.natural_orbitals(D, None, None, orthogonalize_mo = False, n_eps = 0.001)

```

D: ndarray of shape (MO x MO)

```
--> diagonalization of D

n: ndarray of shape (NO)
c: ndarray of shape (MO x NO)
```

---

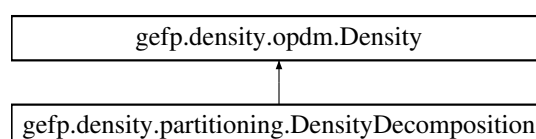
Last Revision: Gundelf

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

- `gefp/gefp/density/opdm.py`

## 18.30 `gefp.density.partitioning.DensityDecomposition` Class Reference

Inheritance diagram for `gefp.density.partitioning.DensityDecomposition`:



### Public Member Functions

- `def __init__ (self, aggregate, method='hf', acbs=True, jk_type='direct', no_cutoff=0.000, xc_scale=1.0, l_dds=True, cc_relax=True, verbose=False, n_eps=5.0E-5, kwargs)`
- `def compute (self, polar_approx=True)`
- `def deformation\_density (self, name)`
- `def compute_monomers (self)`
- `def compute_full_QM (self)`
- `def compute_densities (self)`
- `def compute_coulomb (self)`
- `def compute_pauli (self)`
- `def compute_polar (self)`
- `def compute_polar_approx (self)`
- `def __repr__ (self)`
- `def print_out (self)`
- `def doublet (self, A, B)`
- `def triplet (self, A, B, C)`
- `def matrix_power (self, M, x, eps=1.0e-6)`
- `def rms (self, m1, m2)`

## Public Attributes

- **aggregate**
- **method**
- **data**
- **matrix**
- **vars**
- **xc\_recommended**
- **kwargs**
- **acbs**
- **no\_cutoff**
- **n\_eps**
- **cc\_relax**
- **l\_dds**
- **verbose**
- **xc\_scale**
- **monomers\_computed**
- **densities\_computed**
- **energy\_coulomb\_computed**
- **energy\_pauli\_computed**
- **energy\_polar\_computed**
- **energy\_polar\_approx\_computed**
- **energy\_ind\_computed**
- **energy\_disp\_computed**
- **energy\_ct\_compute**
- **energy\_full\_QM\_computed**
- **dms\_ind\_computed**
- **dms\_disp\_computed**
- **dms\_ct\_computed**
- **bfs**
- **global\_jk**
- **nmo\_t**
- **nbf\_t**

### 18.30.1 Detailed Description

---

Density-Based Decomposition Scheme of Mandado and Hermida-Ramon with partitioning of polarization deformation density into induction, dispersion and charge-transfer contributions.

```
--> DDS <--  
--> Density Decomposition Scheme <--
```

## References:

- \* Mandado and Hermida-Ramon, J. Chem. Theory Comput. 2011, 7, 633-641. (JCTC 2011)
- \* B<sub>l</sub>asiak, J. Chem. Phys. 2018 149 (16), 164115. (JCP 2018)

## Constructor arguments and options:

- o aggregate - Psi4 molecular aggregate with at least two fragments
- o method - QM method (hf, mp2, cc2, ccSD)
- o acbs - use aggregate-centred basis set for calculations of wavefunctions.  
Otherwise use monomer-centred basis sets and composite Hadamard addition of AO spaces. ACBS=False result in no correction for BSSE.
- o jk\_type - type of Psi4 JK object.
- o no\_cutoff - cutoff for natural occupancies threshold. All natural orbitals with occupancies less or equal to the threshold will be neglected.
- o xc\_scale - scaling parameter for exchange-correlation density
- o l\_dds - compute also linear DDS total interaction energy
- o kwargs - additional Psi4-relevant options.

## Usage example:

```
solver = DensityDecomposition(aggr, method='hf',
                              acbs=True,
                              jk_type='direct',
                              no_cutoff=0.000,
                              xc_scale=1.0,
                              l_dds=True,
                              n_eps=5.0E-5,
                              cc_relax=True,
                              verbose=True,
                              **kwargs)

solver.compute(polar_approx=False)

dD_pauli = solver.deformation_density('pau')
dD_pol   = solver.deformation_density('pol')
dD       = solver.deformation_density('fqm')

# dictionaries:
# 1. accessing variables
solver.vars
# 2. accessing aggregate data
solver.matrix
# 3. accessing unperturbed fragment data (expert)
solver.data

print(solver)
solver.print_out()
```

Created : Gundelf  
Last Revision: Gundelf



## 18.30.2 Member Function Documentation

### 18.30.2.1 compute()

```
def gefp.density.partitioning.DensityDecomposition.compute (
    self,
    polar_approx = True )
```

\

Perform the full density and interaction energy decompositions.

Options:

- o polar\_approx - in addition to exact polarization energy, compute also the approximated polarization energy using NO-expansion of exchange-correlation 2-electron density and exact Pauli, polarization and unperturbed 1-electron densities. Default: True

Notes:

- o Exact polarization energy is always computed as a difference between the full QM interaction energy and all the remaining energies (Coulombic, exchange and repulsion energies).

### 18.30.2.2 deformation\_density()

```
def gefp.density.partitioning.DensityDecomposition.deformation_density (
    self,
    name )
```

\

Compute the deformation 1-particle density matrix.  
Returns density matrix in AO basis of entire molecular aggregate.

Possible <name> entries:

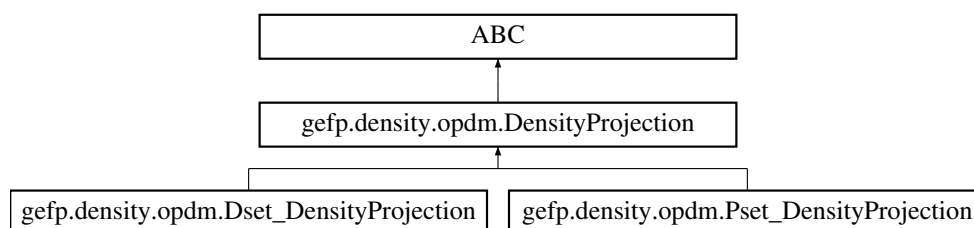
- o fqm - full QM deformation density
- o pau - Pauli-repulsion denformation density
- o pol - polarization deformation density
- o ind - induction part of polarization deformation density
- o dis - dispersion part of polarization deformation density
- o ct - charge-transfer part of polarization deformation density

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

- gefp/gefp/density/partitioning.py

## 18.31 gefp.density.opdm.DensityProjection Class Reference

Inheritance diagram for gefp.density.opdm.DensityProjection:



### Public Member Functions

- `def __init__ (self, np, S)`
- `def compute (self, n, c)`

### Static Public Member Functions

- `def create (np, dtype='p', S=None)`

#### 18.31.1 Detailed Description

\
 Gradient Projection Algorithms.  
 Ref.: Pernal, Cancas, J. Chem. Phys. 2005

Usage:

```
proj = DensityProjection.create(np, dtype='p', S=None)
n, c = proj.compute(n, c, S)
```

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

- `gefp/gefp/density/opdm.py`

## 18.32 gefp.basis.optimize.DFBasis Class Reference

### Public Member Functions

- `def __init__ (self, mol, templ_file='templ.dat', param_file='param.dat', bounds_file=None, constraints=())`
- `def basisset (self, param=None)`
- `def print (self, param=None)`
- `def save (self, out='oepfit.gbs', param=None)`

## Public Attributes

- **mol**
- **templ**
- **param**
- **n\_param**
- **bounds**
- **constraints**
- **basis**

## Static Public Attributes

- float **exp\_lower\_bound** = 0.01
- float **exp\_upper\_bound** = 10000.0
- float **ctr\_lower\_bound** = -2.0
- float **ctr\_upper\_bound** = 2.0

### 18.32.1 Detailed Description

Basis set object to be optimized.

Notes:

- o Default bounds can be modified by resetting static variables  
DFBasis.exp\_lower\_bound  
DFBasis.exp\_upper\_bound  
DFBasis.ctr\_lower\_bound  
DFBasis.ctr\_upper\_bound  
prior to calling DFBasis if not using the driver.gdf\_basis\_optimizer.

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

- gefp/gefp/basis/optimize.py

## 18.33 gefp.basis.optimize.DFBasisOptimizer Class Reference

### Public Member Functions

- def **\_\_init\_\_** (self, oep)
- def **fit** (self, maxiter=1000, tolerance=1e-9, method='slsqp', opt\_global=False, temperature=500, stepsize=500, take\_step=None, accept\_test=None)
- def **compute\_error** (self, basis, rms=False)

## Public Attributes

- **oep**
- **basis\_fit**
- **param**

### 18.33.1 Detailed Description

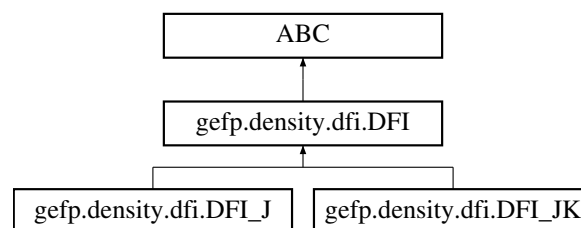
Method that optimizes DF basis set.

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

- `gefp/gefp/basis/optimize.py`

## 18.34 gefp.density.dfi.DFI Class Reference

Inheritance diagram for `gefp.density.dfi.DFI`:



## Public Member Functions

- `def __init__ (self, frags)`
- `def run (self, maxit=100, conv=1.0e-5, verbose_scf=False, conv_scf=1.0e-5, maxit_scf=100, damp_scf=0.14, ndamp_scf=0)`
- `def aggregate (self)`
- `def wfn (self, i)`
- `def epsilon (self, i)`
- `def Cocc (self, i)`
- `def C (self, i)`
- `def D (self, i)`
- `def F (self, i)`
- `def V (self, i)`
- `def E (self, i)`

## Static Public Member Functions

- `def create (frags, j_only=False)`

## Public Attributes

- **enuc**
- **en\_0**

### 18.34.1 Detailed Description

---

Density Fragment Interaction (DFI) Method

---

Demo for SCF-DFI method (closed shells).

Usage:

```
dfi = DFI(fragment_1, fragment_2, [...]) # OR: dfi = DFI(fragments)
dfi.run(maxit=30, conv=1.0e-5, verbose_scf=False, conv_scf=1.0e-5, maxit_scf=100, damp_scf=0.14, ndamp_scf=0)
```

Notes:

- o fragmet\_i is a psi4.core.Molecule wit one Psi4 Fragment
  - o fragments is a psi4.core Molecule with multiple Psi4 Fragments ('--' separator in input)
  - o SCF of unperturbed molecule is solved by Psi4, while the subsequent SCF's in DFI iterations are solved by SCF class instances of this Demo.
- 

Last Revision: Gundolf

### 18.34.2 Member Function Documentation

#### 18.34.2.1 run()

```
def gefp.density.dfi.DFI.run (
    self,
    maxit = 100,
    conv = 1.0e-5,
    verbose_scf = False,
    conv_scf = 1.0e-5,
    maxit_scf = 100,
    damp_scf = 0.14,
    ndamp_scf = 0 )
```

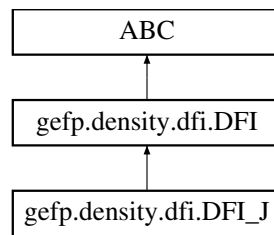
Runs DFI iterations

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

- gefp/gefp/density/dfi.py

## 18.35 gefp.density.dfi.DFI\_J Class Reference

Inheritance diagram for gefp.density.dfi.DFI\_J:



### Public Member Functions

- `def __init__ (self, frags)`

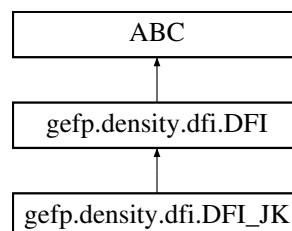
### Additional Inherited Members

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

- `gefp/gefp/density/dfi.py`

## 18.36 gefp.density.dfi.DFI\_JK Class Reference

Inheritance diagram for gefp.density.dfi.DFI\_JK:



### Public Member Functions

- `def __init__ (self, frags)`

### Additional Inherited Members

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

- `gefp/gefp/density/dfi.py`

## 18.37 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)

### 18.37.1 Detailed Description

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.

### 18.37.2 Constructor & Destructor Documentation

#### 18.37.2.1 DIISManager()

```
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

### 18.37.3 Member Function Documentation

#### 18.37.3.1 compute()

```
void oepdev::DIISManager::compute (
    void )
```

Perform DIIS interpolation.

#### 18.37.3.2 put()

```
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

#### 18.37.3.3 update()

```
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.

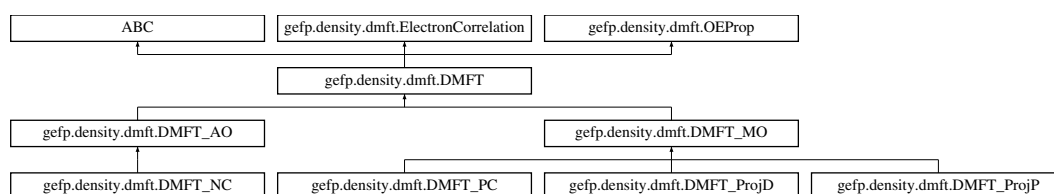
The documentation for this class was generated from the following files:

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

## 18.38 gefp.density.dmft.DMFT Class Reference

Inheritance diagram for gefp.density.dmft.DMFT:





## Public Member Functions

- `def __init__ (self, wfn, xc_functional, v_ext=default_v_ext, guess=default_guess, step_mode=default_step_mode)`
- `def create (cls, wfn, xc_functional=default_xc_functional, v_ext=default_v_ext, guess=default_guess, algorithm=default_algorithm, step_mode=default_step_mode, kwargs)`
- `def run (self, conv=default_convergence, maxit=default_maxiter, verbose=default_verbose_run, g_0=default_g0, g=default_g, restart=False, kwargs)`
- `def set_gradient_mode (self, exact=False, approx=False, num=False)`
- `def E (self)`
- `def D (self)`
- `def Dmo (self)`
- `def Dao (self)`
- `def C (self)`
- `def N (self)`
- `def scalar_correlation (self)`
- `def matrix_correlation (self)`
- `def dipole (self)`
- `def quadrupole (self)`
- `def abbr (self)`

## Static Public Member Functions

- `def name ()`

## Static Public Attributes

- `default_xc_functional = XCFunctional.create('hf')`
- `string default_algorithm = 'proj-p'`
- `float default_convergence = 0.00001`
- `int default_maxiter = 100`
- `bool default_verbose_run = True`
- `float default_g0 = 0.0001`
- `default_v_ext = None`
- `string default_guess = 'hcore'`
- `string default_step_mode = 'search'`
- `float default_g = 0.01`

### 18.38.1 Detailed Description

\

---

The Density Matrix Functional Theory. Abstract Base.

---

Usage:

```
# init
dmft = DMFT.create(wfn,
                   xc_functional = DMFT.default_xc_functional,
                   v_ext         = DMFT.default_v_ext         ,
                   guess          = DMFT.default_guess         ,
                   algorithm       = DMFT.default_algorithm     ,
                   step_mode       = DMFT.default_step_mode     ,
                   **kwargs)

# options
dmft.set_gradient_mode(exact=False, approx=False, num=False)

# run
dmft.run(conv       = DMFT.default_convergence,
          maxit      = DMFT.default_maxiter    ,
          verbose    = DMFT.default_verbose_run,
          g_0        = DMFT.default_g0         ,
          g          = DMFT.default_g         ,
          restart    = False                   ,
          **kwargs):
```

Options (init):

- o wfn - psi4.core.Wavefunction object. Must contain SCF LCAO-MO coefficients
- o xc\_functional - XCFunctional object. Default: HF functional object.
- o v\_ext - External potential in AO basis. Default is no potential.
- o guess - Guess for the density:
  - o 'hcore' - diagonalize Hcore Hamiltonian (default)
  - o 'current' - use the current density matrix stored in 'wfn'
- o algorithm - DMFT algorithm to converge the density matrix.
  - o 'proj-d' - Projected gradient algorithm on D-sets. Suitable only for
  - o 'proj-p' - Projected gradient algorithm on P-sets. Suitable for any
  - o 'nc' - Direct optimization within n and C parameter space. Suitable
  - o 'pc' - Direct optimization within p and C parameter space. Not
- o step\_mode - How to search for next guess: estimate step length in steepest descent
  - o 'search' - Compute from two last density guesses (default)
  - o 'constant' - Apply constant step.

Options (optional setup): relevant only for 'proj-p' algorithm

- o exact - Compute exact derivatives of XC energy wrt P
  - o approx - Compute approximate derivatives of XC energy wrt P
  - o num - Compute numerically derivatives of XC energy wrt P
- If using this, set only one of the above three to True.

Options (run):

- o conv - Energy convergence (default 0.00001)
- o maxit - Maximum number of iterations (default 100)
- o verbose - Print detailed information or not (default True)
- o g\_0 - Initial SD step size (default 0.0001)

- o `g` - Constant SD step size (default 0.01)
  - o `restart` - Wheather to restart the calculations or not (default False)
- 

## 18.38.2 Member Function Documentation

### 18.38.2.1 `create()`

```
def gefp.density.dmft.DMFT.create (
    cls,
    wfn,
    xc_functional = default_xc_functional,
    v_ext = default_v_ext,
    guess = default_guess,
    algorithm = default_algorithm,
    step_mode = default_step_mode,
    kwargs )

\
    Create DMFT solver.
```

### 18.38.2.2 `run()`

```
def gefp.density.dmft.DMFT.run (
    self,
    conv = default_convergence,
    maxit = default_maxiter,
    verbose = default_verbose_run,
    g_0 = default_g0,
    g = default_g,
    restart = False,
    kwargs )

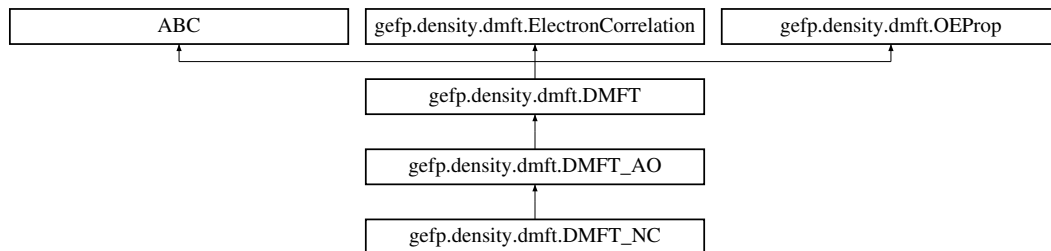
\
    Run the DMFT calculations.
```

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

- `gefp/gefp/density/dmft.py`

## 18.39 gefp.density.dmft.DMFT\_AO Class Reference

Inheritance diagram for gefp.density.dmft.DMFT\_AO:



### Public Member Functions

- `def __init__ (self, wfn, xc_functional, v_ext, guess, step)`
- `def dipole (self)`
- `def Dmo (self)`

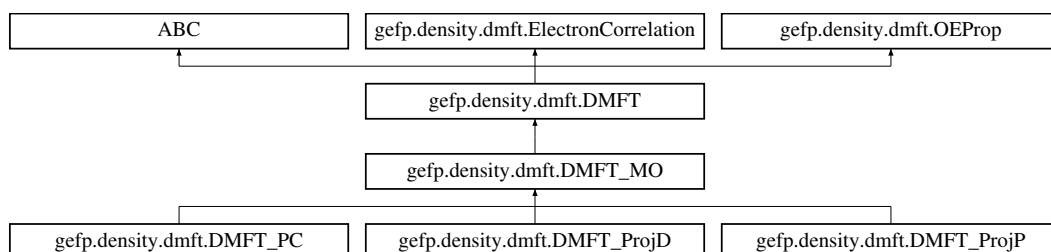
### Additional Inherited Members

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

- `gefp/gefp/density/dmft.py`

## 18.40 gefp.density.dmft.DMFT\_MO Class Reference

Inheritance diagram for gefp.density.dmft.DMFT\_MO:



### Public Member Functions

- `def __init__ (self, wfn, xc_functional, v_ext, guess, step)`

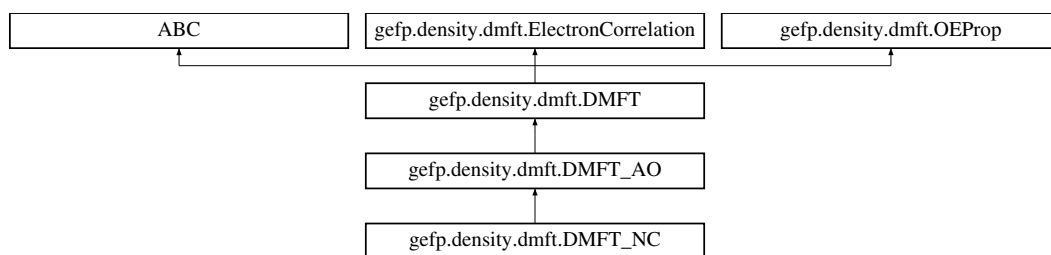
## Additional Inherited Members

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

- gefp/gefp/density/dmft.py

## 18.41 gefp.density.dmft.DMFT\_NC Class Reference

Inheritance diagram for gefp.density.dmft.DMFT\_NC:



## Public Member Functions

- def **\_\_init\_\_** (self, wfn, xc\_functional, v\_ext, guess, step)
- def **abbr** (self)

## Static Public Member Functions

- def **name** ()

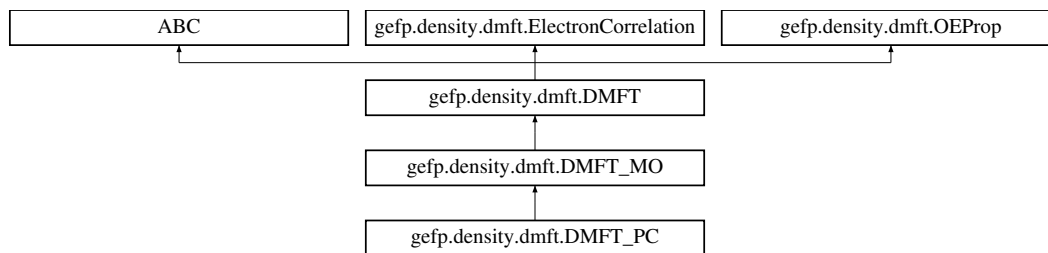
## Additional Inherited Members

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

- gefp/gefp/density/dmft.py

## 18.42 gefp.density.dmft.DMFT\_PC Class Reference

Inheritance diagram for gefp.density.dmft.DMFT\_PC:



### Public Member Functions

- `def __init__ (self, wfn, xc_functional, v_ext, guess, step)`
- `def abbr (self)`

### Static Public Member Functions

- `def name ()`

### Public Attributes

- `g`

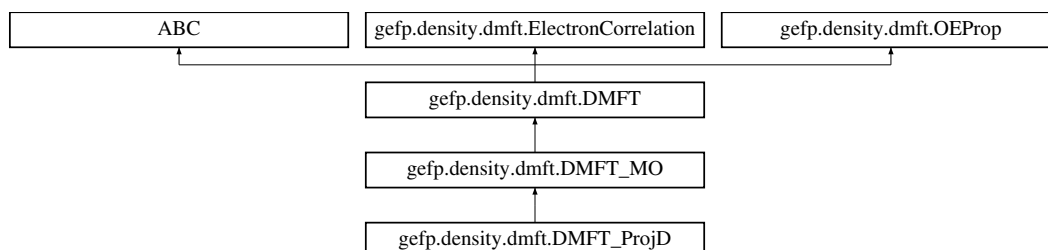
### Additional Inherited Members

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

- `gefp/gefp/density/dmft.py`

## 18.43 gefp.density.dmft.DMFT\_ProjD Class Reference

Inheritance diagram for gefp.density.dmft.DMFT\_ProjD:



### Public Member Functions

- `def __init__ (self, wfn, xc_functional, v_ext, guess, step)`
- `def abbr (self)`

## Static Public Member Functions

- def **name** ()

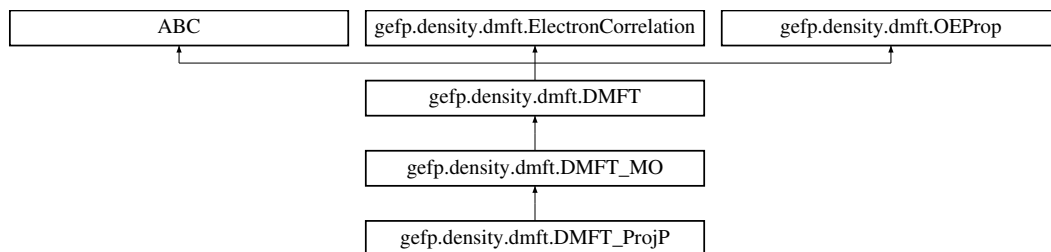
## Additional Inherited Members

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

- gefp/gefp/density/dmft.py

## 18.44 gefp.density.dmft.DMFT\_ProjP Class Reference

Inheritance diagram for gefp.density.dmft.DMFT\_ProjP:



## Public Member Functions

- def **\_\_init\_\_** (self, wfn, xc\_functional, v\_ext, guess, step)
- def **abbr** (self)

## Static Public Member Functions

- def **name** ()

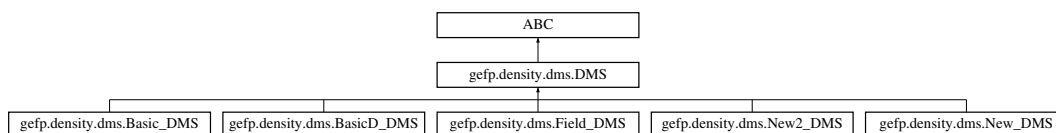
## Additional Inherited Members

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

- gefp/gefp/density/dmft.py

## 18.45 gefp.density.dms.DMS Class Reference

Inheritance diagram for gefp.density.dms.DMS:



## Public Member Functions

- `def __init__ (self, typ)`
- `def create (cls, dms_type='da', order_type='basic')`
- `def available_orders (self)`
- `def N (self)`
- `def n (self)`
- `def M (self)`
- `def B (self, m, n)`
- `def set_bfs (self, bfs)`
- `def set_M (self, M)`
- `def set_s1 (self, s)`
- `def set_s2 (self, s)`
- `def set_ca (self, Ca_occ, Ca_vir)`
- `def Ca_occ (self)`
- `def Ca_vir (self)`
- `def M (self)`
- `def B (self, m, n)`
- `def read (self, out)`
- `def write (self, out)`

### 18.45.1 Detailed Description

Density Matrix Susceptibility Tensor.  
Abstract Base.

This describes the DMS of Density or Fock matrix as EFP.

EFPs:

- nuclear structure
- unperturbed Density or Fock matrix (alpha or beta)
- DMS tensors

Functionalities:

- rotation, translation and superimposition
- read/write capabilities based on FRG format (Solvshift)

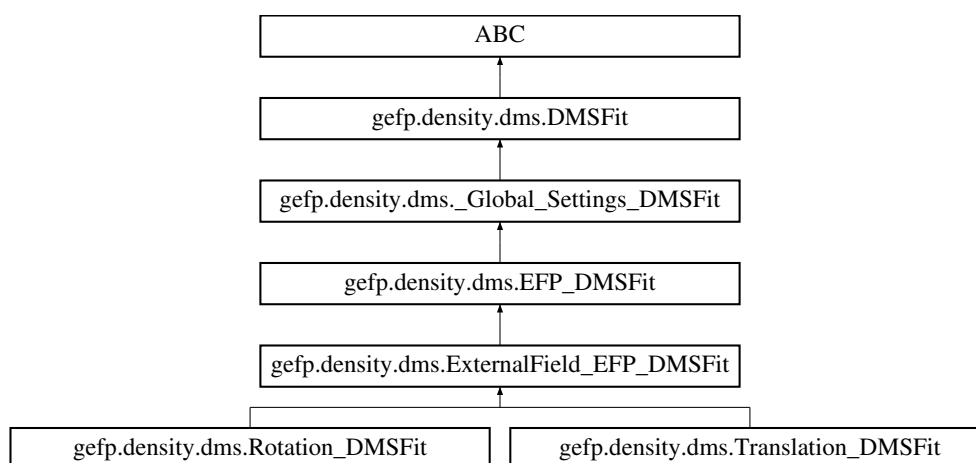
The documentation for this class was generated from the following file:

- `gefp/gefp/density/dms.py`



## 18.46 gefp.density.dms.DMSFit Class Reference

Inheritance diagram for gefp.density.dms.DMSFit:



### Public Member Functions

- `def __init__ (self, mol, method, nsamples, dms_types, order_type, use_iterative_model, use_external_field_model)`
- `def create (cls, mol, fit_type="transl", dms_types="da, g, order_type='basic', nsamples=100, method='scf', use_iterative_model=True, use_external_field_model=False)`
- `def run (self)`
- `def B (self)`

### Static Public Attributes

- float **minimum\_atom\_atom\_distance** = 1.2 / psi4.constants.bohr2angstroms
- float **minimum\_atom\_charge\_distance** = 4.5
- float **srange** = 4.0
- bool **stop\_when\_error\_hessian** = True
- bool **compute\_dmatpol\_susceptibilities** = False
- bool **generate\_random\_samples** = True
- bool **read\_samples** = False
- int **dmatpol\_ntest\_charge** = 40
- float **dmatpol\_test\_charge** = 0.05
- float **dmatpol\_esp\_pad\_shpere** = 6.0
- int **dmatpol\_gradient\_rank** = 0
- bool **replace\_extfield\_with\_dmatpol** = False

### 18.46.1 Detailed Description

Method to fit DMS tensors.

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

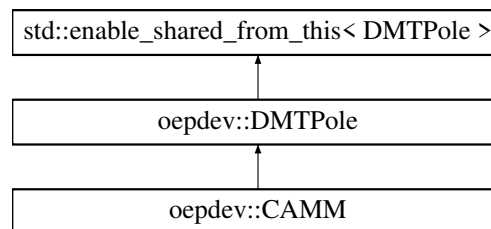
- `gefp/gefp/density/dms.py`

## 18.47 oepdev::DMTPole Class Reference

Distributed Multipole Analysis Container and Computer. Abstract Base.

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

Inheritance diagram for oepdev::DMTPole:



## Public Member Functions

### Accessors

- 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 psi::SharedVector [centre](#) (int x) const  
*Get the position of the \*x\*th distribution centre.*
- virtual psi::SharedVector [origin](#) (int x) const  
*Get the position of the \*x\*th distribution origin.*
- 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 ith distribution.*
- virtual psi::SharedMatrix **dipoles** (int i) const
- Get the distributed dipoles for the ith distribution.*
- virtual psi::SharedMatrix **quadrupoles** (int i) const
- Get the distributed quadrupoles for the ith distribution.*
- virtual psi::SharedMatrix **octupoles** (int i) const
- Get the distributed octupoles for the ith distribution.*
- virtual psi::SharedMatrix **hexadecapoles** (int i) const
- Get the distributed hexadecapoles for the ith distribution.*
- virtual int **n\_sites** () const
- Get the number of distributed sites.*
- virtual int **n\_dmtp** () const
- Get the number of distributions.*

## Mutators

- 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 ith distribution.*
- void **set\_dipoles** (psi::SharedMatrix M, int i)
- Set the distributed dipoles for the ith distribution.*
- void **set\_quadrupoles** (psi::SharedMatrix M, int i)
- Set the distributed quadrupoles for the ith distribution.*
- void **set\_octupoles** (psi::SharedMatrix M, int i)
- Set the distributed octupoles for the ith distribution.*
- void **set\_hexadecapoles** (psi::SharedMatrix M, int i)
- Set the distributed hexadecapoles for the ith distribution.*

## Transformators

- virtual void [recenter](#) (psi::SharedMatrix new\_origins)  
*Change origins of the distributed multipole moments of all sets.*
- void [translate](#) (psi::SharedVector transl)  
*Translate the DMTP sets.*
- void [rotate](#) (psi::SharedMatrix rotmat)  
*Rotate the DMTP sets.*
- double [superimpose](#) (psi::SharedMatrix ref\_xyz, std::vector< int > suplist={})  
*Superimpose the DMTP sets.*

## Computers

- void [compute](#) (std::vector< psi::SharedMatrix > D, std::vector< bool > t)  
*Compute DMTP's from the set of the one-particle density matrices.*
- void [compute](#) (void)  
*Compute ground state DMTP.*
- 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](#) (const double &x, const double &y, const double &z, [MultipoleConvergence::ConvergenceLevel](#) max\_clevel=[MultipoleConvergence::R5](#))  
*Evaluate the generalized potential at a given point.*
- std::shared\_ptr< [MultipoleConvergence](#) > [field](#) (const double &x, const double &y, const double &z, [MultipoleConvergence::ConvergenceLevel](#) max\_clevel=[MultipoleConvergence::R5](#))  
*Evaluate the generalized field at a given point.*

## Printers

- virtual void [print\\_header](#) () const  
*Print the header.*
- void [print](#) () const  
*Print the contents.*

## Static Public Member Functions

- static [MultipoleConvergence::ConvergenceLevel](#) [determine\\_dmtp\\_convergence\\_level](#) (const std::string &option)

## Protected Member Functions

### Protected Interface

- [DMTPole](#) (std::shared\_ptr< psi::Wavefunction > wfn, int n)  
*Construct an empty DMTP object from the wavefunction.*
- virtual void [compute](#) (psi::SharedMatrix D, bool transition, int i)

- Compute DMTP's from the one-particle density matrix.*
- void `compute_integrals` ()  
*Compute multipole integrals.*
- void `compute_order` ()  
*Compute maximum order of the integrals.*
- virtual void `recenter` (psi::SharedMatrix new\_origins, int i)  
*Change origins of the distributed multipole moments of ith set.*
- virtual void `allocate` ()  
*Initialize and allocate memory.*
- virtual void `copy_from` (const DMTPole \*)  
*Deep-copy the matrix and DMTP data.*

## Protected Attributes

### Basic

- 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)*
- std::vector< psi::SharedMatrix > `mplnts_`  
*Multipole integrals.*

### Sizing

- int `nDMTPs_`  
*Number of DMTP's.*
- int `nSites_`  
*Number of DMTP sites.*
- int `order_`  
*Maximum order of the multipole.*

### Descriptors

- 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?*

## Geometry

- psi::SharedMatrix [centres\\_](#)  
*DMTP centres.*
- psi::SharedMatrix [origins\\_](#)  
*DMTP origins.*

## Multipoles

- 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](#)

## Constructors and Destructor

- 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.*
- static std::shared\_ptr< [DMTPole](#) > [empty](#) (void)  
*Build an empty DMTP object of no type.*
- [DMTPole](#) (void)  
*Construct an empty DMTP object of no type.*
- [DMTPole](#) (const [DMTPole](#) \*)  
*Copy constructor.*
- std::shared\_ptr< [DMTPole](#) > [clone](#) (void) const  
*Make a deep copy (wfn\_, mol\_, and primary\_ are shallow-copied)*
- virtual [~DMTPole](#) ()  
*Destructor.*

### 18.47.1 Detailed Description

Handles the distributed multipole expansions up to hexadecapoles. Distributed centres as well as DMTP origins are allowed to be located in arbitrary points in space. The object describes a set of  $N$  DMTP's, that can be generated by providing one-particle density matrices in AO basis. Nuclear contributions can be switched on or off separately for each DMTP within a set. The following operations on the DMTP sets are available through the API:

- translation
- rotation
- superimposition
- recentering the origins
- computing the generalized property from another DMTP set

See also

[MultipoleConvergence](#)

### 18.47.2 Constructor & Destructor Documentation

#### 18.47.2.1 DMTPole() [1/2]

```
oepdev::DMTPole::DMTPole (
    void )
```

Do not use this constructor. Use the [DMTPole::empty](#) method.

#### 18.47.2.2 DMTPole() [2/2]

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

Parameters

<i>wfn</i>	- wavefunction
<i>n</i>	- number of DMTP sets

Do not use this constructor. Use the [DMTPole::build](#) method.

## 18.47.3 Member Function Documentation

### 18.47.3.1 build()

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

#### 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

### 18.47.3.2 compute() [1/2]

```
void oepdev::DMTPole::compute (
    std::vector< psi::SharedMatrix > D,
    std::vector< bool > t )
```

#### Parameters

<i>D</i>	- list of one-particle density matrices
<i>t</i>	- list of flags determining if density is of transition type or not

### 18.47.3.3 compute() [2/2]

```
void oepdev::DMTPole::compute (
    void )
```

Compute DMTP's from the *sum* of the ground-state alpha and beta one-particle density matrices (t=false, i=0). Results in a usual DMTP analysis of a molecule's charge density distribution.



## 18.47.3.4 determine\_dmtp\_convergence\_level()

```
MultipoleConvergence::ConvergenceLevel oepdev::DMTPole::determine_dmtp_convergence_level
(
    const std::string & option ) [static]
```

Determine the CAMM convergence for a given global option

## Parameters

<i>option</i>	- string for option
---------------	---------------------

## 18.47.3.5 empty()

```
std::shared_ptr< DMTPole > oepdev::DMTPole::empty (
    void ) [static]
```

## Returns

Blank DMTP distribution with memory allocated by no data.

## 18.47.3.6 energy()

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

## 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.

### 18.47.3.7 field()

```
std::shared_ptr< MultipoleConvergence > oepdev::DMTPole::field (
    const double & x,
    const double & y,
    const double & z,
    MultipoleConvergence::ConvergenceLevel max_clevel = MultipoleConvergence::R5
)
```

#### Parameters

<i>x</i>	- location x-th Cartesian component
<i>y</i>	- location y-th Cartesian component
<i>z</i>	- location z-th Cartesian component
<i>max_clevel</i>	- maximum convergence level (see below).

#### Returns

The generalized field 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.

### 18.47.3.8 potential()

```
std::shared_ptr< MultipoleConvergence > oepdev::DMTPole::potential (
    const double & x,
    const double & y,
    const double & z,
    MultipoleConvergence::ConvergenceLevel max_clevel = MultipoleConvergence::R5
)
```

**Parameters**

<i>x</i>	- location x-th Cartesian component
<i>y</i>	- location y-th Cartesian component
<i>z</i>	- location z-th Cartesian component
<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.

**18.47.3.9 recenter()**

```
void oepdev::DMTPole::recenter (
    psi::SharedMatrix new_origins ) [virtual]
```

**Parameters**

<i>new_origins</i>	- matrix with coordinates of the new origins $\{\mathbf{r}_{\text{new}}\}$ .
--------------------	--

**Note**

The number of origins has to be equal to the number of distributed centres.

Recentering of the multipoles affects the distributed dipoles and higher moments. The moments are given as

$$\begin{aligned}
 q_{\text{new}} &= q_{\text{old}} \\
 \mu_{\text{new}} &= \mu_{\text{old}} - q_{\text{old}} \Delta^{(1)} \\
 \Theta_{\text{new}} &= \Theta_{\text{old}} + q_{\text{old}} \Delta^{(2)} - \sum_{\mathcal{P}_2} \mathcal{P}_2 \left[ (q_{\text{old}} \mathbf{r}_{\text{old}} + \mu_{\text{old}}) \otimes \Delta^{(1)} \right] \\
 \Omega_{\text{new}} &= \Omega_{\text{old}} - q_{\text{old}} \Delta^{(3)} + \sum_{\mathcal{P}_3} \mathcal{P}_3 \left[ (q_{\text{old}} \mathbf{r}_{\text{old}} + \mu_{\text{old}}) \otimes \Delta^{(2)} \right] - \sum_{\mathcal{P}_6} \mathcal{P}_6 \left[ (q_{\text{old}} \mathbf{r}_{\text{old}}^2 + \mu_{\text{old}} \otimes \mathbf{r}_{\text{old}} + \Theta_{\text{old}}) \otimes \Delta^{(1)} \right] \\
 \Xi_{\text{new}} &= \Xi_{\text{old}} + q_{\text{old}} \Delta^{(4)} - \sum_{\mathcal{P}_3} \mathcal{P}_3 \left[ (q_{\text{old}} \mathbf{r}_{\text{old}} + \mu_{\text{old}}) \otimes \Delta^{(3)} \right] + \sum_{\mathcal{P}_3} \mathcal{P}_3 \left[ (q_{\text{old}} \mathbf{r}_{\text{old}}^2 + \mu_{\text{old}} \otimes \mathbf{r}_{\text{old}} + \Theta_{\text{old}}) \otimes \Delta^{(2)} \right] - \sum_{\mathcal{P}_6} \mathcal{P}_6 \left[ (q_{\text{old}} \mathbf{r}_{\text{old}}^3 + \mu_{\text{old}} \otimes \mathbf{r}_{\text{old}}^2 + \Theta_{\text{old}} \otimes \mathbf{r}_{\text{old}} + \Omega_{\text{old}}) \otimes \Delta^{(1)} \right]
 \end{aligned}$$

where

$$\begin{aligned}
 \Delta^{(1)} &\equiv \mathbf{r}_{\text{new}} - \mathbf{r}_{\text{old}} \\
 \Delta^{(2)} &\equiv \mathbf{r}_{\text{new}}^2 - \mathbf{r}_{\text{old}}^2 \\
 \Delta^{(3)} &\equiv \mathbf{r}_{\text{new}}^3 - \mathbf{r}_{\text{old}}^3 \\
 \Delta^{(4)} &\equiv \mathbf{r}_{\text{new}}^4 - \mathbf{r}_{\text{old}}^4
 \end{aligned}$$

In the above equations, the distributed centre label was omitted (redundant) as each distributed site of multipoles is independent of the others. TODO - Finish for octupoles and hexadecapoles!  
 -> define the permutation operators!

**18.47.3.10 rotate()**

```
void oepdev::DMTPole::rotate (
    psi::SharedMatrix rotmat )
```

**Parameters**

<i>rotmat</i>	- Cartesian rotation matrix $\mathbf{r}$
---------------	--

Centers and origins, as well as dipole, quadrupole, octupole and hexadecapole moments are

transformed according to:

$$\begin{aligned}
 x_a^{(i)} &\rightarrow \sum_{a'} x_{a'}^{(i)} r_{a'a} \\
 o_a^{(i)} &\rightarrow \sum_{a'} o_{a'}^{(i)} r_{a'a} \\
 \mu_a^{(i)} &\rightarrow \sum_{a'} \mu_{a'}^{(i)} r_{a'a} \\
 \Theta_a^{(i)} &\rightarrow \sum_{a'b'} \Theta_{a'b'}^{(i)} r_{a'a} r_{b'b} \\
 \Omega_a^{(i)} &\rightarrow \sum_{a'b'c'} \Omega_{a'b'c'}^{(i)} r_{a'a} r_{b'b} r_{c'c} \\
 \Xi_a^{(i)} &\rightarrow \sum_{a'b'c'd'} \Xi_{a'b'c'd'}^{(i)} r_{a'a} r_{b'b} r_{c'c} r_{d'd}
 \end{aligned}$$

where the definition of  $r_{a'a}$  is consistent with the Kabsch algorithm implemented in [KabschSuperimposer](#).

See also

[KabschSuperimposer](#)

### 18.47.3.11 superimpose()

```
double oepdev::DMTPole::superimpose (
    psi::SharedMatrix ref_xyz,
    std::vector< int > suplist = {} )
```

#### Parameters

<i>ref_xyz</i>	- target geometry to superimpose
<i>suplist</i>	- superimposition list

#### Returns

the RMS of superimposition Kabsch algorithm is used for superimposition.

See also

[KabschSuperimposer](#)

## 18.47.4 Friends And Related Function Documentation

### 18.47.4.1 MultipoleConvergence

```
friend class MultipoleConvergence [friend]
```

Convergence of multipole moment series.

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

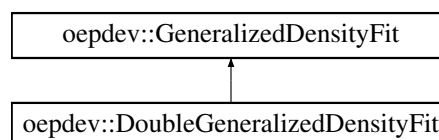
- [oepdev/lib3d/dmtp.h](#)
- [oepdev/lib3d/dmtp\\_base.cc](#)

## 18.48 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

#### 18.48.1 Detailed Description

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.

#### 18.48.2 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  $\hat{v}$ .

### 18.48.2.1 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.

### 18.48.3 Member Function Documentation

#### 18.48.3.1 compute()

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

#### Returns

The OEP coefficients  $G_{\xi i}$

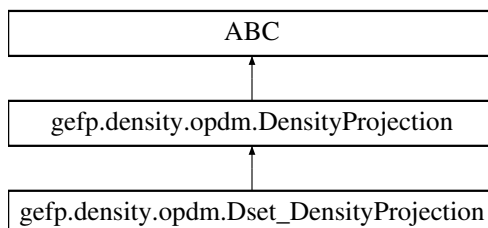
Implements [oepdev::GeneralizedDensityFit](#).

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

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

## 18.49 gefp.density.opdm.Dset\_DensityProjection Class Reference

Inheritance diagram for gefp.density.opdm.Dset\_DensityProjection:



#### Public Member Functions

- `def __init__ (self, np, S)`



## Additional Inherited Members

### 18.49.1 Detailed Description

\

Gradient Projection Algorithm on D-sets.  
 Ref.: Pernal, Cancas, J. Chem. Phys. 2005

Notes:

- o Appropriate only for HF functional.

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

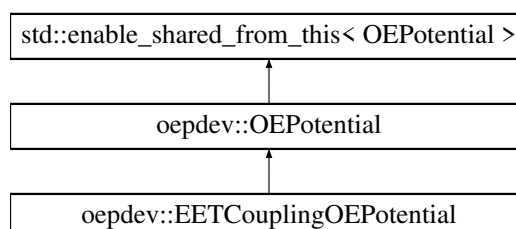
- gefp/gefp/density/opdm.py

## 18.50 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  
*Header information.*

## Additional Inherited Members

### 18.50.1 Detailed Description

Contains the following OEP types:

- `Fujimoto.GDF` - Joint OEP type for ET(L), ET(HL), HT(H) and HT(HL)
- `Fujimoto.CIS` - CIS data
- `Fujimoto.EXCH`- Pure-exchange coupling matrix  $G_{\mu\nu} \equiv (\mu\mu|vv)$
- `Fujimoto.CT_M`- ( $HH|LL$ ) integral for the H\_34 Hamiltonian matrix elements (CT)

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

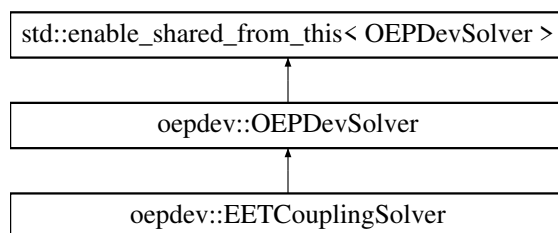
- `oepdev/liboep/oep.h`
- `oepdev/liboep/oep_coupling_eet.cc`

## 18.51 oepdev::EETCouplingSolver Class Reference

Compute the EET coupling energy between unperturbed wavefunctions.

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

Inheritance diagram for `oepdev::EETCouplingSolver`:



## Public Member Functions

- **EETCouplingSolver** (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

### 18.51.1 Detailed Description

The implemented methods are shown below

Table 18.32: Methods available in the Solver

Keyword	Method Description
<b>Benchmark Methods</b>	
FUJIMOTO_TI_CIS	<i>Default.</i> EET Coupling by Fujimoto JPC 2012.
<b>OEP-Based Methods</b>	
FUJIMOTO_TI_CIS	<i>Default.</i> OEP-based TI/CIS expressions.

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

### TI/CIS Method (Fujimoto JPC 2012).

In the simplest version of TI/CIS approach, the Hamiltonian of the molecular aggregate (dimer) is constructed from the CIS approximation and 4 basis functions constructed as follows:

$$\begin{aligned} |\Phi_1\rangle &= |\Psi_A^{(e)} \otimes \Psi_B^{(g)}\rangle \\ |\Phi_2\rangle &= |\Psi_A^{(g)} \otimes \Psi_B^{(e)}\rangle \\ |\Phi_3\rangle &= |\Psi_A^{(+)} \otimes \Psi_B^{(-)}\rangle \\ |\Phi_4\rangle &= |\Psi_A^{(-)} \otimes \Psi_B^{(+)}\rangle \end{aligned}$$

where *g* and *e* superscripts denote the ground and excited state of a molecule, + and – label the cationic and anionic state, respectively, whereas  $|\Psi_X \otimes \Psi_Y\rangle$  denotes the antisymmetrized Hartree product of the monomer wavefunctions. The associated diagonal Hamiltonian matrix

elements can be defined as

$$\begin{aligned}
 \langle \Phi_1 | \mathcal{H} - E_0 | \Phi_1 \rangle &\equiv E_1 = E_{e \rightarrow g}^A + \sum_{\mu \nu \in A} \left( P_{\nu \mu}^{A(e)} - P_{\nu \mu}^{A(g)} \right) \times \left\{ V_{\mu \nu}^{B(\text{nuc})} + \sum_{\lambda \sigma \in B} P_{\lambda \sigma}^{B(g)} \left[ (\mu \nu | \sigma \lambda) - \frac{1}{2} (\mu \lambda | \sigma \nu) \right] \right\} \\
 \langle \Phi_2 | \mathcal{H} - E_0 | \Phi_2 \rangle &\equiv E_2 = E_{e \rightarrow g}^B + \sum_{\mu \nu \in B} \left( P_{\nu \mu}^{B(e)} - P_{\nu \mu}^{B(g)} \right) \times \left\{ V_{\mu \nu}^{A(\text{nuc})} + \sum_{\lambda \sigma \in A} P_{\lambda \sigma}^{A(g)} \left[ (\mu \nu | \sigma \lambda) - \frac{1}{2} (\mu \lambda | \sigma \nu) \right] \right\} \\
 \langle \Phi_3 | \mathcal{H} - E_0 | \Phi_3 \rangle &\equiv E_3 = -\varepsilon_H^A + \varepsilon_L^B - (H^A H^A | L^B L^B) \\
 \langle \Phi_4 | \mathcal{H} - E_0 | \Phi_4 \rangle &\equiv E_4 = \varepsilon_L^A - \varepsilon_H^B - (L^A L^A | H^B H^B)
 \end{aligned}$$

The associated off-diagonal Hamiltonian matrix elements can be defined as

$$\begin{aligned}
 \langle \Phi_1 | \mathcal{H} | \Phi_2 \rangle &\equiv V^{\text{Coul}} + V^{\text{Exch}} + V^{\text{Ovrl}} \\
 \langle \Phi_1 | \mathcal{H} | \Phi_3 \rangle &\equiv V^{\text{ET1}} \\
 \langle \Phi_2 | \mathcal{H} | \Phi_4 \rangle &\equiv V^{\text{ET2}} \\
 \langle \Phi_1 | \mathcal{H} | \Phi_4 \rangle &\equiv V^{\text{HT1}} \\
 \langle \Phi_2 | \mathcal{H} | \Phi_3 \rangle &\equiv V^{\text{HT2}} \\
 \langle \Phi_3 | \mathcal{H} | \Phi_4 \rangle &\equiv V^{\text{CT}}
 \end{aligned}$$

where the Forster-type Coulombic (Coul), Dexter-type exchange (Exch), remaining overlap correction (Ovrl), as well as the electron, hole and charge (ET, HT, CT) transfer contributions are defined. The exchange-Coulomb coupling takes the form

$$\begin{aligned}
 V^{\text{Coul}} &= \frac{V^{\text{Coul},(0)}}{1 - S_{12}^2} \\
 V^{\text{Exch}} &= \frac{V^{\text{Exch},(0)}}{1 - S_{12}^2} \\
 V^{\text{Ovrl}} &= -\frac{(E_1 + E_2)S_{12}}{2(1 - S_{12}^2)}
 \end{aligned}$$

The overlap-corrected ET, HT and CT matrix elements read

$$\begin{aligned}
 V^{\text{ET1}} &= [1 - S_{13}^2]^{-1} \left\{ V^{\text{ET1},(0)} - \frac{1}{2} (E_1 + E_2) S_{13} \right\} \\
 V^{\text{ET2}} &= [1 - S_{24}^2]^{-1} \left\{ V^{\text{ET2},(0)} - \frac{1}{2} (E_1 + E_2) S_{24} \right\} \\
 V^{\text{HT1}} &= [1 - S_{14}^2]^{-1} \left\{ V^{\text{HT1},(0)} - \frac{1}{2} (E_1 + E_2) S_{14} \right\} \\
 V^{\text{HT2}} &= [1 - S_{23}^2]^{-1} \left\{ V^{\text{HT2},(0)} - \frac{1}{2} (E_1 + E_2) S_{23} \right\} \\
 V^{\text{CT}} &= [1 - S_{34}^2]^{-1} \left\{ V^{\text{CT},(0)} - \frac{1}{2} (E_1 + E_2) S_{34} \right\}
 \end{aligned}$$

In the above equations, the superscript (0) denotes that the matrix elements are not affected by the overlap between molecular wavefunctions, and are given by

$$\begin{aligned}
 V^{\text{Coul},(0)} &= \sum_{\mu\nu\in A} \sum_{\lambda\sigma\in B} P_{\nu\mu}^{g\rightarrow e(A)} P_{\lambda\sigma}^{g\rightarrow e(B)} (\mu\nu|\sigma\lambda) \\
 V^{\text{Exch},(0)} &= -\frac{1}{2} \sum_{\mu\nu\in A} \sum_{\lambda\sigma\in B} P_{\nu\mu}^{g\rightarrow e(A)} P_{\lambda\sigma}^{g\rightarrow e(B)} (\mu\lambda|\sigma\nu) \\
 V^{\text{ET1},(0)} &= t_{H\rightarrow L}^A \left\{ (L^A|\mathcal{F}|L^B) + 2(L^A H^A|H^A L^B) - (L^A L^B|H^A H^A) \right\} \\
 V^{\text{ET2},(0)} &= t_{H\rightarrow L}^B \left\{ (L^A|\mathcal{F}|L^B) + 2(L^A H^B|H^B L^B) - (L^A L^B|H^B H^B) \right\} \\
 V^{\text{HT1},(0)} &= t_{H\rightarrow L}^A \left\{ -(H^A|\mathcal{F}|H^B) + 2(H^A L^A|L^A H^B) - (H^A H^B|L^A L^A) \right\} \\
 V^{\text{HT2},(0)} &= t_{H\rightarrow L}^B \left\{ -(H^A|\mathcal{F}|H^B) + 2(H^A L^B|L^B H^B) - (H^A H^B|L^B L^B) \right\} \\
 V^{\text{CT},(0)} &= 2(H^A L^B|L^A H^B) - (H^A H^B|L^A L^B)
 \end{aligned}$$

In the above,  $\mathcal{F}$  is the Fock operator whereas  $H$  and  $L$  denote the HOMO and LUMO orbitals, respectively. The overlap integrals between the basis states are approximated by

$$\begin{aligned}
 S_{12} &\equiv (\Phi_1|\Phi_2) \cong -\frac{1}{N_{el}^{AB}} \text{Tr} \left[ \mathbf{P}^{g\rightarrow e(A)} \mathbf{s}^{AB} \mathbf{P}^{g\rightarrow e(B)} \mathbf{s}^{BA} \right] \\
 S_{13} &\equiv (\Phi_1|\Phi_3) \cong -\frac{t_{H\rightarrow L}^A}{N_{el}^{AB}} S_{LL}^{AB} \\
 S_{14} &\equiv (\Phi_1|\Phi_4) \cong +\frac{t_{H\rightarrow L}^A}{N_{el}^{AB}} S_{HH}^{AB} \\
 S_{24} &\equiv (\Phi_2|\Phi_4) \cong -\frac{t_{H\rightarrow L}^B}{N_{el}^{AB}} S_{LL}^{AB} \\
 S_{23} &\equiv (\Phi_2|\Phi_3) \cong +\frac{t_{H\rightarrow L}^B}{N_{el}^{AB}} S_{HH}^{AB} \\
 S_{34} &\equiv (\Phi_3|\Phi_4) \cong -\frac{1}{N_{el}^{AB}} S_{HH}^{AB} S_{LL}^{AB}
 \end{aligned}$$

where the overlap between molecular orbitals  $U$  and  $W$  is given by

$$S_{UW}^{AB} \equiv \mathbf{s}^{AB} : \mathbf{c}_U^A \otimes \mathbf{c}_W^B$$

and  $\mathbf{s}^{AB}$  is the AO overlap matrix between molecule A and B atomic basis functions.

For a closed-shell system, the EET coupling constant for two electronic transitions can be given approximately by

$$V \approx V^{\text{Direct}} + V^{\text{Indirect}}$$

where the overlap-corrected direct and indirect coupling constants are

$$\begin{aligned}
 V^{\text{Direct}} &= V^{\text{Coul}} + V^{\text{Exch}} + V^{\text{Ovrl}} \\
 V^{\text{Indirect}} &= V^{\text{TI-2}} + V^{\text{TI-3}}
 \end{aligned}$$

with

$$V^{\text{TI}-2} = -\frac{V^{\text{ET1}}V^{\text{HT2}}}{E_3 - E_1} - \frac{V^{\text{ET2}}V^{\text{HT1}}}{E_4 - E_1}$$

$$V^{\text{TI}-3} = \frac{V^{\text{CT}}(V^{\text{ET1}}V^{\text{ET2}} + V^{\text{HT1}}V^{\text{HT2}})}{(E_3 - E_1)(E_4 - E_1)}$$

### Fock matrix in AB space

In the current implementation, Fock matrix in the AB space, that is necessary to evaluate ET and HT matrix elements, can be defined as

1. the AB block of full Hartree-Fock SCF Fock matrix for entire system;
2. the zeroth-order Fock matrix that is composed of monomer's unperturbed ground-state 1-particle density matrices.

In the latter case, the Fock matrix in AO representation is given by:

$$F_{\alpha\in A, \beta\in B}^{AB} \approx T_{\alpha\beta} + V_{\alpha\beta}^{A(\text{nuc})} + V_{\alpha\beta}^{B(\text{nuc})} + \sum_{\mu\nu\in A} P_{\nu\mu}^{A(g)} G_{\alpha\beta, \mu\nu} + \sum_{\sigma\lambda\in B} P_{\lambda\sigma}^{B(g)} G_{\alpha\beta, \sigma\lambda}$$

where

$$G_{\alpha\beta, \gamma\delta} \equiv (\alpha\beta|\gamma\delta) - \frac{1}{2}(\alpha\delta|\gamma\beta)$$

### Mulliken approximated exchange-like contributions.

Exchange and CT contributions require ERI's of type (AB,AB). It is instructive to approximate these contributions in terms of the Coulomb-like ERI's for the sake of testing of OEP-based approximations which are given in the next Section.

Application of the Mullipen approximation

$$(ij|kl) \approx \frac{1}{4} S_{ij} S_{kl} [(ii|kk) + (jj|kk) + (ii|ll) + (jj|ll)]$$

results in the following approximations to the exchange-like terms

$$V^{\text{Exch},(0)} \approx -\frac{1}{8} \sum_{\mu\nu\in A} \sum_{\lambda\sigma\in B} P_{\nu\mu}^{g\rightarrow e(A)} P_{\lambda\sigma}^{g\rightarrow e(B)} S_{\mu\lambda} S_{\sigma\nu} [(\mu\mu|\sigma\sigma) + (\lambda\lambda|\nu\nu) + (\mu\mu|\nu\nu) + (\lambda\lambda|\sigma\sigma)]$$

$$V^{\text{CT},(0)} \approx \frac{1}{2} S_{HL}^{AB} S_{LH}^{AB} [r_{HL}^A + r_{HL}^B + \rho_H^A \odot \rho_H^B + \rho_L^A \odot \rho_L^B]$$

$$- \frac{1}{4} S_{HH}^{AB} S_{LL}^{AB} [r_{HL}^A + r_{HL}^B + \rho_H^A \odot \rho_L^B + \rho_L^A \odot \rho_H^B]$$

The former can be rewritten in a more convenient to implement formula:

$$V^{\text{Exch},(0)} \approx -\frac{1}{4} \sum_{\mu\in A} \sum_{\nu\in B} (\mu\mu|\sigma\sigma) [\mathbf{P}^A \mathbf{s}^{AB}]_{\mu\sigma} [\mathbf{P}^B \mathbf{s}^{BA}]_{\sigma\mu} - \frac{1}{8} \sum_{\mu\nu\in A} P_{\nu\mu}^A (\mu\mu|\nu\nu) [\mathbf{s}^{AB} \mathbf{P}^B \mathbf{s}^{BA}]_{\mu\nu} - \frac{1}{8} \sum_{\sigma\lambda\in B} P_{\lambda\sigma}^B (\lambda\lambda|\sigma\sigma)$$

In the CT term,

$$\begin{aligned} r_{HL}^A &\equiv \rho_H^A \odot \rho_L^A \\ r_{HL}^B &\equiv \rho_H^B \odot \rho_L^B \end{aligned}$$

where the effective Coulombic interaction energies are defined by

$$\rho_U^A \odot \rho_W^B \equiv (U^A U^A | W^A W^A)$$

## OEP-Based Methods

TODO

### OEP-Based TI/CIS theory

After introducing OEP's, the original TI/CIS theory by Fujimoto is reformulated *without* approximation as TODO

## 18.51.2 Member Function Documentation

### 18.51.2.1 compute\_benchmark()

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

Each solver object has one DEFAULT benchmark method

#### Parameters

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

Implements [oepdev::OEPDevSolver](#).

### 18.51.2.2 compute\_oep\_based()

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

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:

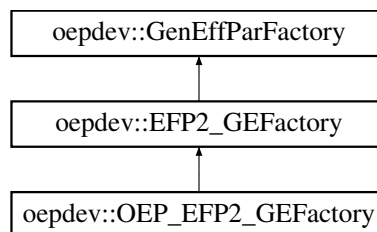
- oepdev/libsolver/[solver.h](#)
- oepdev/libsolver/solver\_coupling\_eet.cc

## 18.52 oepdev::EFP2\_GEFactory Class Reference

EFP2 GEFP Factory.

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

Inheritance diagram for oepdev::EFP2\_GEFactory:



### Public Member Functions

- [EFP2\\_GEFactory](#) (std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)  
*Construct from Psi4 options.*
- virtual [~EFP2\\_GEFactory](#) ()  
*Destruct.*
- virtual std::shared\_ptr< [GenEffPar](#) > [compute](#) (void)  
*Compute the EFP2 parameters.*

### Protected Member Functions

- virtual std::shared\_ptr< [oepdev::DMTPole](#) > [compute\\_dmt](#) (void)
- virtual void [compute\\_lmoc](#) (void)
- virtual std::shared\_ptr< [oepdev::CPHF](#) > [compute\\_cphf](#) (void)
- virtual void [assemble\\_parameters](#) (void)
- virtual void [assemble\\_geometry\\_data](#) (void)
- virtual void [assemble\\_dmt\\_data](#) (void)
- virtual void [assemble\\_lmo\\_centroids](#) (void)

- virtual void **assemble\_fock\_matrix** (void)
- virtual void **assemble\_distributed\_polarizabilities** (void)

## Protected Attributes

- std::shared\_ptr< [oepdev::GenEffPar](#) > **EFP2Parameters\_**

## Additional Inherited Members

### 18.52.1 Detailed Description

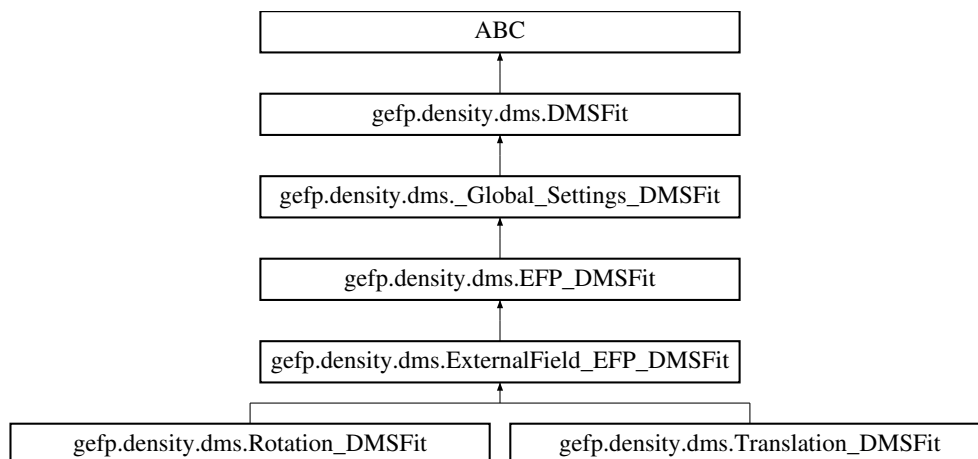
Basic interface for the EFP2 parameters.

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

- [oepdev/libgefp/gefp.h](#)
- [oepdev/libgefp/gefp\\_efp2.cc](#)

## 18.53 `gefp.density.dms.EFP_DMSFit` Class Reference

Inheritance diagram for `gefp.density.dms.EFP_DMSFit`:



## Public Member Functions

- def **\_\_init\_\_** (self, mol, method, nsamples, dms\_types, order\_type, use\_iterative\_model, use\_external\_field\_model)
- def **run** (self)
- def **B** (self, m, n, dtype='da')

## Additional Inherited Members

### 18.53.1 Detailed Description

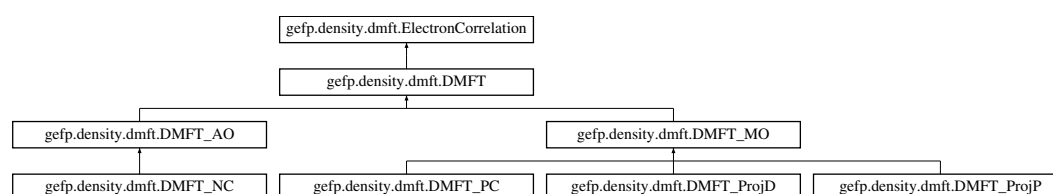
DMS Fitting for EFP-like Molecular Fragments

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

- `gefp/gefp/density/dms.py`

## 18.54 gefp.density.dmft.ElectronCorrelation Class Reference

Inheritance diagram for gefp.density.dmft.ElectronCorrelation:



## Static Public Member Functions

- `def degree_of_correlation (dmft, scalar=True)`

### 18.54.1 Detailed Description

\  
The Electron Correlation: Dynamic and Non-dynamic Correlation.

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

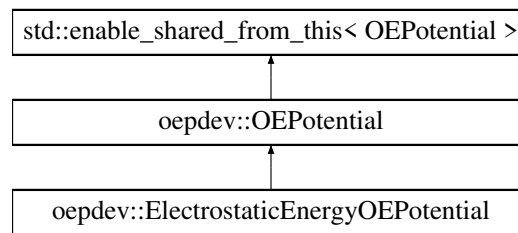
- `gefp/gefp/density/dmft.py`

## 18.55 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  
*Header information.*

## Additional Inherited Members

### 18.55.1 Detailed Description

Contains the following OEP types:

- $V$

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

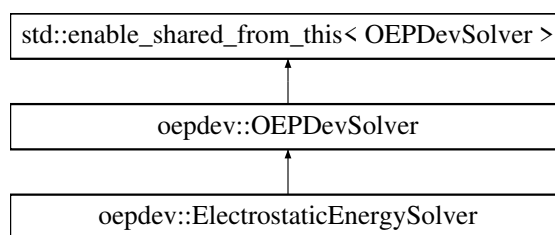
- oepdev/liboep/[oep.h](#)
- oepdev/liboep/oep\_energy\_coul.cc

## 18.56 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

### 18.56.1 Detailed Description

The implemented methods are shown in below

Table 18.35: Methods available in the Solver

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

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,

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

**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

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

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

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

and the electron-electron repulsion energy is

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

## OEP-Based Methods

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

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

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

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

*Notes:*

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

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

for reference purposes.

- In order to construct this solver, **always** use the [OEPDevSolver::build](#) static factory method.

## 18.56.2 Member Function Documentation

### 18.56.2.1 compute\_benchmark()

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

Each solver object has one DEFAULT benchmark method

**Parameters**

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

Implements [oepdev::OEPDevSolver](#).

### 18.56.2.2 compute\_oep\_based()

```
double ElectrostaticEnergySolver::compute_oep_based (
```

```
const std::string & method = "DEFAULT" ) [virtual]
```

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:

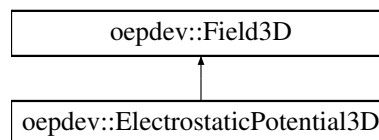
- [oepdev/libsolver/solver.h](#)
- [oepdev/libsolver/solver\\_energy\\_coul.cc](#)

## 18.57 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



### 18.57.1 Detailed Description

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

## 18.58 gefp.core.driver.Entry Class Reference

### Public Member Functions

- `def __init__ (self, pade, description_short, description_full)`

### Public Attributes

- `pade`
- `description_short`
- `description_full`

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

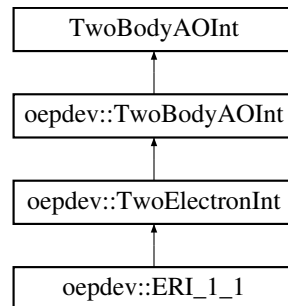
- `gefp/gefp/core/driver.py`

## 18.59 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)*

### 18.59.1 Detailed Description

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

For each  $(n_1, l_1, m_1) \in P$  :  
 For each  $(n_2, l_2, m_2) \in Q$  :  

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

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

## 18.59.2 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:

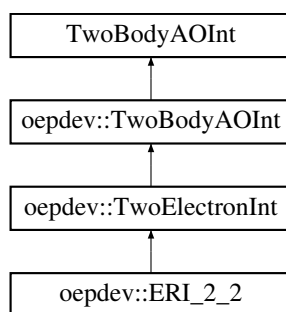
- oepdev/libints/[eri.h](#)
- oepdev/libints/eri.cc

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

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

```
#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)*

### 18.60.1 Detailed Description

ERI's are computed for a shell quartet (PQ|RS) 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  $(n_4, l_4, m_4) \in S$  :  
 $\text{ERI} = (AB|CD)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}]$

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

### 18.60.2 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

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

In the above equation, the multiplicative constants are given as

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

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

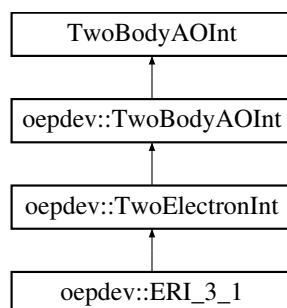
- oepdev/libints/[eri.h](#)
- oepdev/libints/eri.cc

## 18.61 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)*

### 18.61.1 Detailed Description

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](#).

### 18.61.2 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](#)

## 18.62 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.*

### 18.62.1 Detailed Description

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

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

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 fluorium atom. Default: 2.3 [A.U.]
- `ESP_VDW_RADIUS_CL` - The vdW radius for chlorium atom. Default: 2.9 [A.U.]

**18.62.2 Constructor & Destructor Documentation****18.62.2.1 ESPSolver() [1/2]**

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

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



## Parameters

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

18.62.2.2 `ESPSolver()` [2/2]

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

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

## Parameters

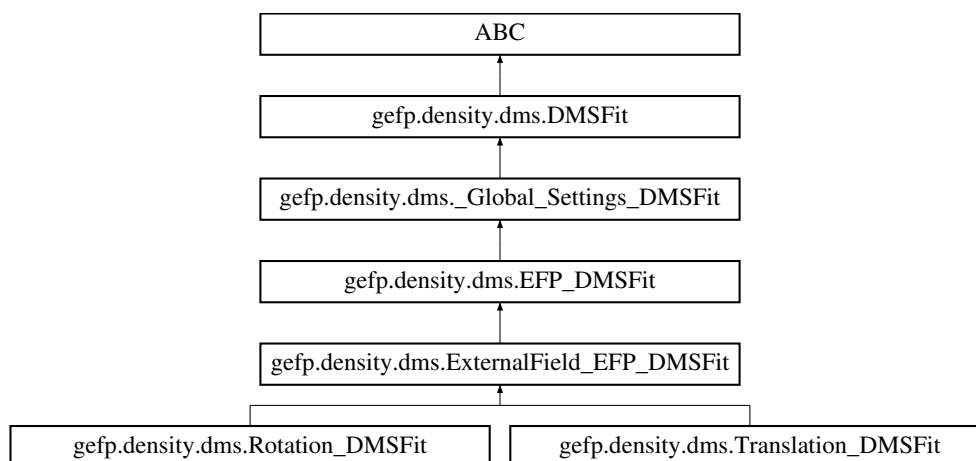
<i>field</i>	- oepdev 3D vector field object
<i>centres</i>	- matrix with coordinates of charge distribution centres

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

- `oepdev/lib3d/esp.h`
- `oepdev/lib3d/esp.cc`

18.63 `gefp.density.dms.ExternalField_EFP_DMSFit` Class Reference

Inheritance diagram for `gefp.density.dms.ExternalField_EFP_DMSFit`:



## Public Member Functions

- `def __init__ (self, mol, method, nsamples, dms_types, order_type, use_iterative_model, use_external_field_model)`

## Additional Inherited Members

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

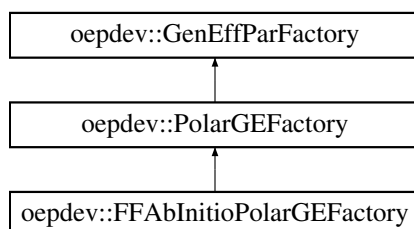
- `gefp/gefp/density/dms.py`

## 18.64 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

### 18.64.1 Detailed Description

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

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

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

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

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:

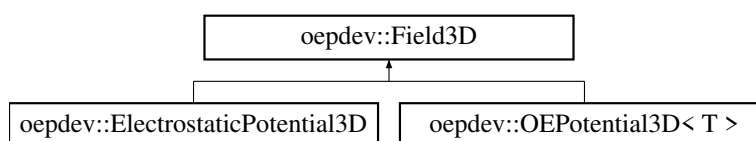
- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp\_polar\_ffabinitio.cc

## 18.65 oepdev::Field3D Class Reference

General Vector Dield 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 &nz, 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?*

### 18.65.1 Detailed Description

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`

### 18.65.2 Constructor & Destructor Documentation

### 18.65.2.1 Field3D()

```
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 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);

## 18.65.3 Member Function Documentation

### 18.65.3.1 build() [1/2]

```
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]
```

#### 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

### 18.65.3.2 build() [2/2]

```
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]

```

### Parameters

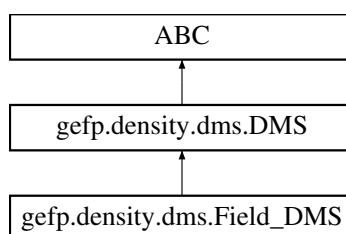
<i>ndim</i>	- dimensionality of 3D field (1: scalar field, >2: vector field)
<i>type</i>	- type of 3D field
<i>nx</i>	- number of points along x direction
<i>ny</i>	- number of points along y direction
<i>nz</i>	- number of points along z direction
<i>px</i>	- padding distance along x direction
<i>py</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

## 18.66 gefp.density.dms.Field\_DMS Class Reference

Inheritance diagram for gefp.density.dms.Field\_DMS:



### Public Member Functions

- `def __init__ (self, type='da')`

### 18.66.1 Detailed Description

Basic model of DMS that handles induction up to second-order in the external electric field without any other electronic densities.

The order of DMS blocks:

Block -----	DMS order -----	Interaction -----	Symmetry -----	Dimension -----	Group -----
1.	(1,0)	Induction	Symmetric	(n,n,N,3)	Z(1)
2.	(2,0)	Induction	Symmetric	(n,n,N,3,3)	Z(1)

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

- `gefp/gefp/density/dms.py`

## 18.67 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**

### 18.67.1 Detailed Description

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

- `oepdev/libutil/unitary_optimizer.h`

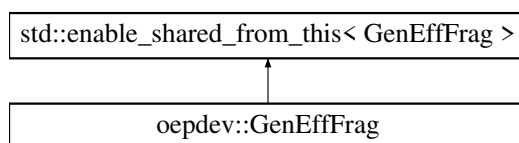
## 18.68 oepdev::GenEffFrag Class Reference

Generalized Effective Fragment. Container Class.

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



Inheritance diagram for oepdev::GenEffFrag:



## Public Member Functions

### Constructors and Destructor

- [GenEffFrag](#) ()  
*Initialize with default name of GEFP (Default)*
- [GenEffFrag](#) (std::string name)  
*Initialize with custom name of GEFP.*
- [GenEffFrag](#) (const [GenEffFrag](#) \*)  
*Copy Constructor.*
- std::shared\_ptr< [GenEffFrag](#) > [clone](#) (void) const  
*Make a deep copy.*
- [~GenEffFrag](#) ()  
*Destruct.*

### Transformators

- 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 [superimpose](#) (psi::SharedMolecule targetMol, std::vector< int > supList)  
*Superimpose.*
- void [superimpose](#) (void)  
*Superimpose to the structure held in frag\_*

### Mutators

- void [set\\_ndocc](#) (int n)  
*Set the number of doubly occupied MOs.*
- void [set\\_nbf](#) (int n)  
*Set the number of primary basis functions.*
- void [set\\_molecule](#) (const psi::SharedMolecule mol)  
*Set the fragment molecule.*
- void [set\\_basisset](#) (std::string key, psi::SharedBasisSet basis)  
*Set the basis set.*
- 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.*

## Accessors

- int [nbf](#) (void) const

*Grab the number of primary basis functions.*

- int [natom](#) (void) const

*Grab the number of atoms.*

- int [ndocc](#) (void) const

*Grab the number of doubly occupied molecular orbitals.*

- psi::SharedMolecule [molecule](#) (void) const

*Grab the molecule attached to this fragment.*

- 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

### Parameters

- std::map< std::string, std::shared\_ptr< [GenEffPar](#) > > [parameters](#)

*Dictionary of All GEF Parameters.*

- std::map< std::string, psi::SharedBasisSet > [basissets](#)

*Dictionary of All Basis Sets.*

## Protected Member Functions

### Interface Computers

- double [compute\\_pairwise\\_energy](#) (std::string theory, std::shared\_ptr< [GenEffFrag](#) > other)

- double **compute\_pairwise\_energy\_efp2\_coul** (std::shared\_ptr< [GenEffFrag](#) > other)
- double **compute\_pairwise\_energy\_efp2\_exrep** (std::shared\_ptr< [GenEffFrag](#) > other)
- double **compute\_pairwise\_energy\_efp2\_ind** (std::shared\_ptr< [GenEffFrag](#) > other)
- double **compute\_pairwise\_energy\_efp2\_ct** (std::shared\_ptr< [GenEffFrag](#) > other)
- double **compute\_pairwise\_energy\_efp2\_disp** (std::shared\_ptr< [GenEffFrag](#) > other)
- double **compute\_pairwise\_energy\_oep\_efp2\_exrep** (std::shared\_ptr< [GenEffFrag](#) > other)
- double **compute\_pairwise\_energy\_oep\_efp2\_ct** (std::shared\_ptr< [GenEffFrag](#) > other)

## Protected Attributes

- std::string [name\\_](#)  
*Name of GEFP.*
- psi::SharedMolecule [frag\\_](#)  
*Structure.*
- int [nbf\\_](#)  
*Number of primary basis functions.*
- int [natom\\_](#)  
*Number of atoms.*
- int [ndocc\\_](#)  
*Number of doubly occupied MOs.*
- std::shared\_ptr< [GenEffPar](#) > **densityMatrixSusceptibilityGEF\_**

## Computers

- double [energy](#) (std::string theory, std::shared\_ptr< [GenEffFrag](#) > other)  
*Compute interaction energy between this and other fragment.*
- static double [compute\\_energy](#) (std::string theory, std::vector< std::shared\_ptr< [GenEffFrag](#) >> fragments, bool manybody)  
*Compute interaction energy in a cluster of fragments.*
- static double [compute\\_many\\_body\\_energy](#) (std::string theory, std::vector< std::shared\_ptr< [GenEffFrag](#) >> fragments)  
*Compute interaction energy in a cluster of fragments by using manybody routine.*

### 18.68.1 Detailed Description

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

See also

[GenEffPar](#), [GenEffParFactory](#)

## 18.68.2 Member Function Documentation

### 18.68.2.1 compute\_energy()

```
double oepdev::GenEffFrag::compute_energy (
    std::string theory,
    std::vector< std::shared_ptr< GenEffFrag >> fragments,
    bool manybody ) [static]
```

#### Parameters

<i>theory</i>	- theory used to compute energy
<i>fragments</i>	- list of fragments in the system
<i>manybody</i>	- use the manybody routine? If not, pairwise routine is utilized.

#### Returns

interaction energy in [A.U.]

### 18.68.2.2 compute\_many\_body\_energy()

```
double oepdev::GenEffFrag::compute_many_body_energy (
    std::string theory,
    std::vector< std::shared_ptr< GenEffFrag >> fragments ) [static]
```

#### Parameters

<i>theory</i>	- theory used to compute energy
<i>fragments</i>	- list of fragments in the system

#### Returns

interaction energy in [A.U.]

### 18.68.2.3 energy()

```
double oepdev::GenEffFrag::energy (
    std::string theory,
    std::shared_ptr< GenEffFrag > other )
```

## Parameters

<i>theory</i>	- theory used to compute energy
<i>other</i>	- other fragment

## Returns

interaction energy in [A.U.]

## 18.68.2.4 susceptibility() [1/3]

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

## 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
<i>x</i>	- id of the composite Cartesian component

## 18.68.2.5 susceptibility() [2/3]

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

## 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

### 18.68.2.6 susceptibility() [3/3]

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

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

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

## 18.69 oepdev::GenEffPar Class Reference

Generalized Effective Fragment Parameters. Container Class.

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

### Public Member Functions

#### Transformators

- void [rotate](#) (psi::SharedMatrix R)  
*Rotate the parameters in 3D Euclidean space.*
- void [translate](#) (psi::SharedVector t)  
*Translate the parameters in 3D Euclidean space.*
- void [superimpose](#) (psi::SharedMatrix targetXYZ, std::vector< int > supList)  
*Superimpose the parameters in 3D Euclidean space onto a target geometry.*

#### Mutators

- void [set\\_vector](#) (std::string key, psi::SharedVector mat)  
*Set the vector data.*
- void [set\\_matrix](#) (std::string key, psi::SharedMatrix mat)  
*Set the matrix data.*
- void [set\\_dmtip](#) (std::string key, std::shared\_ptr< oepdev::DMTPole > mat)  
*Set the DMTP data.*
- void [set\\_dpole](#) (std::string key, std::vector< psi::SharedMatrix > mats)  
*Set the DPOL data.*
- void [set\\_basiset](#) (std::string key, psi::SharedBasisSet basis)  
*Set the basis set data.*

- 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.*

### Allocators

- 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.*

### Descriptors

- std::string [type](#) () const  
*Type of Parameters.*
- std::string [name](#) () const  
*Name of Parameters.*
- bool [hasDensityMatrixDipolePolarizability](#) () const  
*Does it has dipole polarizability DMS?*
- bool [hasDensityMatrixDipoleDipoleHyperpolarizability](#) () const  
*Does it has dipole-dipole hyperpolarizability DMS?*
- bool [hasDensityMatrixQuadrupolePolarizability](#) () const  
*Does it has quadrupole polarizability DMS?*

### Accessors

- psi::SharedVector [vector](#) (std::string key) const  
*Get the vector data.*
- psi::SharedMatrix [matrix](#) (std::string key) const  
*Get the matrix data.*
- std::shared\_ptr< [oepdev::DMTPole](#) > [dmtp](#) (std::string key) const  
*Get the DMTP data.*

- `std::vector< psi::SharedMatrix > dpol (std::string key) const`  
*Get the DPOL data.*
- `psi::SharedBasisSet basisset (std::string key) const`  
*Get the basis set data.*
- `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.*

## DMS Computers

- `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

### Qualifiers

*Compute the interaction energy between this and other EFP2 fragment.*

### Parameters

par	- other parameters object
-----	---------------------------

- std::string [name\\_](#)  
*The Name of Parameter.*
- std::string [type\\_](#)  
*The Type of Parameter.*
- bool [hasDensityMatrixDipolePolarizability\\_](#)  
*The Name of Parameter.*
- bool [hasDensityMatrixDipoleDipoleHyperpolarizability\\_](#)  
*The Name of Parameter.*
- bool [hasDensityMatrixQuadrupolePolarizability\\_](#)  
*The Name of Parameter.*

## Matrices and Multipoles

- std::vector< std::shared\_ptr< psi::Vector > > [distributedCentres\\_](#)  
*The Positions of the Distributed Centres.*
- std::map< std::string, psi::SharedVector > [data\\_vector\\_](#)  
*Data for Vector Types by Keyword.*
- std::map< std::string, psi::SharedMatrix > [data\\_matrix\\_](#)  
*Data for Matrix Types by Keyword.*
- std::map< std::string, std::shared\_ptr< oepdev::DMTPole > > [data\\_dmtpl\\_](#)  
*Data for DMTP Types by Keyword.*
- std::map< std::string, std::vector< psi::SharedMatrix > > [data\\_dpole\\_](#)  
*Data for DMTP Types by Keyword.*
- std::map< std::string, psi::SharedBasisSet > [data\\_basisset\\_](#)  
*Data for AO Basis Set by Keyword.*

## Density Matrix Susceptibility

- `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.*

## Constructor and Destructor

- [GenEffPar](#) (std::string *name*)  
*Create with name of this parameter.*
- [GenEffPar](#) (const [GenEffPar](#) \*)  
*Copy Constructor.*
- `std::shared_ptr< GenEffPar >` [clone](#) (void) const  
*Make a deep copy.*
- [~GenEffPar](#) ()  
*Destruct.*
- virtual void [copy\\_from](#) (const [GenEffPar](#) \*)  
*Deep-copy the matrix and DMTP data.*

## 18.69.1 Detailed Description

See also

[GenEffFrag](#), [GenEffParFactory](#)

## 18.69.2 Member Function Documentation

### 18.69.2.1 `allocate()`

```
void oepdev::GenEffPar::allocate (
    int fieldRank,
    int fieldGradientRank,
    int nsites,
    int nbF ) [inline]
```

#### Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field $\mathbf{F}$
------------------	--

## Parameters

<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  $\nabla \otimes \mathbf{F}$

18.69.2.2 `basisset()`

```
psi::SharedBasisSet oepdev::GenEffPar::basisset (
    std::string key ) const [inline]
```

## Parameters

<i>key</i>	- keyword for a basis set
------------	---------------------------

## Returns

basis set data type

18.69.2.3 `compute_density_matrix()` [1/4]

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

## Parameters

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

18.69.2.4 `compute_density_matrix()` [2/4]

```
std::shared_ptr< psi::Matrix > oepdev::GenEffPar::compute_density_matrix (
    double fx,
```

```
double fy,
double fz )
```

#### 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.)

#### 18.69.2.5 compute\_density\_matrix() [3/4]

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

#### Parameters

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

#### 18.69.2.6 compute\_density\_matrix() [4/4]

```
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 )
```

#### 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

#### 18.69.2.7 dmtip()

```
std::shared_ptr<oepdev::DMTPole> oepdev::GenEffPar::dmtip (
    std::string key ) const [inline]
```

#### Parameters

<i>key</i>	- keyword for a DMTP
------------	----------------------

**Returns**

DMTP data type

**18.69.2.8 dpol()**

```
std::vector<psi::SharedMatrix> oepdev::GenEffPar::dpol (  
    std::string key ) const [inline]
```

**Parameters**

<i>key</i>	- keyword for a DPOL
------------	----------------------

**Returns**

DPOL data type

**18.69.2.9 matrix()**

```
psi::SharedMatrix oepdev::GenEffPar::matrix (  
    std::string key ) const [inline]
```

**Parameters**

<i>key</i>	- keyword for a matrix
------------	------------------------

**Returns**

matrix data type

**18.69.2.10 rotate()**

```
void oepdev::GenEffPar::rotate (  
    psi::SharedMatrix R )
```

**Parameters**

$R$	- the rotation matrix
-----	-----------------------

**18.69.2.11 set\_basisset()**

```
void oepdev::GenEffPar::set_basisset (
    std::string key,
    psi::SharedBasisSet basis ) [inline]
```

**Parameters**

<i>key</i>	- keyword for a matrix
<i>mat</i>	- matrix

This sets the item in the map `data_basisset_`.

**18.69.2.12 set\_dmtip()**

```
void oepdev::GenEffPar::set_dmtip (
    std::string key,
    std::shared_ptr< oepdev::DMTPole > mat ) [inline]
```

**Parameters**

<i>key</i>	- keyword for a DMTP
<i>dmtip</i>	- DMTP object

This sets the item in the map `data_dmtip_`.

**18.69.2.13 set\_dpil()**

```
void oepdev::GenEffPar::set_dpil (
    std::string key,
    std::vector< psi::SharedMatrix > mats ) [inline]
```

**Parameters**

<i>key</i>	- keyword for a DPOL
<i>dmtip</i>	- DPOL object

This sets the item in the map `data_dpil_`.

**18.69.2.14 set\_matrix()**

```
void oepdev::GenEffPar::set_matrix (
    std::string key,
    psi::SharedMatrix mat ) [inline]
```

## Parameters

<i>key</i>	- keyword for a matrix
<i>mat</i>	- matrix

This sets the item in the map `data_matrix_`.

18.69.2.15 `set_susceptibility()`

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

## 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>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  $\nabla \otimes \mathbf{F}$

18.69.2.16 `set_vector()`

```
void oepdev::GenEffPar::set_vector (
    std::string key,
    psi::SharedVector mat ) [inline]
```

## Parameters

<i>key</i>	- keyword for a vector
<i>mat</i>	- vector

This sets the item in the map `data_vector_`.

**18.69.2.17 superimpose()**

```
void oepdev::GenEffPar::superimpose (
    psi::SharedMatrix targetXYZ,
    std::vector< int > supList )
```

**Parameters**

<i>targetXYZ</i>	- the target geometry
<i>suplist</i>	- the superimposition list

**18.69.2.18 susceptibility()** [1/3]

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

**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
<i>x</i>	- 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.

**18.69.2.19 susceptibility()** [2/3]

```
std::vector<std::shared_ptr<psi::Matrix> > oepdev::GenEffPar::susceptibility
(
```



```

    int fieldRank,
    int fieldGradientRank,
    int i ) const [inline]

```

**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).

**18.69.2.20 susceptibility()** [3/3]

```

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

```

**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  $\nabla \otimes \mathbf{F}$

**18.69.2.21 translate()**

```

void oepdev::GenEffPar::translate (
    psi::SharedVector t )

```

## Parameters

$t$	- the translation vector
-----	--------------------------

## 18.69.2.22 vector()

```
psi::SharedVector oepdev::GenEffPar::vector (
    std::string key ) const [inline]
```

## Parameters

<i>key</i>	- keyword for a vector
------------	------------------------

## Returns

vector data type

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

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

## 18.70 oepdev::GenEffParFactory Class Reference

Generalized Effective Fragment Factory. Abstract Base.

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

Inheritance diagram for oepdev::GenEffParFactory:



## Public Member Functions

## Executor of the Factory

- virtual std::shared\_ptr< [GenEffPar](#) > [compute](#) (void)=0  
*Compute the fragment parameters.*

## Accessors

- virtual std::shared\_ptr< psi::Wavefunction > [wfn](#) (void) const

- *Grab wavefunction.*
- virtual psi::Options & options (void) const  
*Grab options.*
- std::shared\_ptr< oepdev::CPHF > cphf\_solver () const  
*Grab the CPHF object.*
- std::shared\_ptr< oepdev::DMTPole > dmtp () const  
*Grab the DMTP object.*

## Protected Attributes

### Basic data

- std::shared\_ptr< psi::Wavefunction > wfn\_  
*Wavefunction.*
- psi::Options & options\_  
*Psi4 Options.*
- const int nbf\_  
*Number of basis functions.*

### Padding of box

- double cx\_  
*Centre-of-mass coordinates.*
- double cy\_  
*Centre-of-mass coordinates.*
- double cz\_  
*Centre-of-mass coordinates.*
- double radius\_  
*Radius of padding sphere around the molecule.*

### Container objects

- std::shared\_ptr< oepdev::CPHF > cphfSolver\_  
*The CPHF object.*
- std::shared\_ptr< oepdev::DMTPole > dmtp\_  
*The DMTP object.*

### Other Factories

- std::shared\_ptr< oepdev::GenEffParFactory > abInitioPolarizationSusceptibilitiesFactory\_  
*Ab initio polarization susceptibility factory.*

## Constructors and Desctructor

- static std::shared\_ptr< [GenEffParFactory](#) > [build](#) (const std::string &type, std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)  
*Build Density Matrix Susceptibility Generalized Factory.*
- [GenEffParFactory](#) (std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)  
*Construct from wavefunction and Psi4 options.*
- virtual [~GenEffParFactory](#) ()  
*Destruct.*

## Random number generation

- std::default\_random\_engine [randomNumberGenerator\\_](#)  
*Draw random number.*
- std::uniform\_real\_distribution< double > [randomDistribution\\_](#)  
*Draw random number.*
- virtual double [random.double](#) ()  
*Draw random number.*
- virtual std::shared\_ptr< psi::Vector > [draw\\_random\\_point](#) ()  
*Draw random point in 3D space, excluding the vdW region.*

## Van der Waals region

- std::shared\_ptr< psi::Matrix > [excludeSpheres\\_](#)  
*Matrix with vdW sphere information.*
- std::map< std::string, double > [vdwRadius\\_](#)  
*Map with vdW radii.*
- virtual bool [is\\_in\\_vdWsphere](#) (double x, double y, double z) const  
*Is the point inside a vdW region?*

### 18.70.1 Detailed Description

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

See also

[GenEffPar](#), [GenEffFrag](#)

### 18.70.2 Member Function Documentation

## 18.70.2.1 build()

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

## 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 subtype 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:

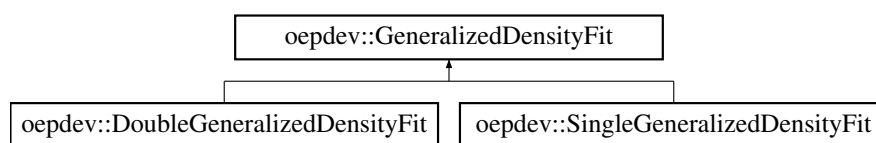
- [oepdev/libgefp/gefp.h](#)
- [oepdev/libgefp/gefp.cc](#)

## 18.71 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 ( $\xi|i\rangle$ ).*
- `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.*

### 18.71.1 Detailed Description

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.

### 18.71.2 Member Function Documentation

**18.71.2.1 build()** [1/2]

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

**Parameters**

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

**Returns**

Generalized Density Fit Computer.

**18.71.2.2 build()** [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]
```

**Parameters**

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

**Returns**

Generalized Density Fit Computer.

**18.71.2.3 compute()**

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

**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:



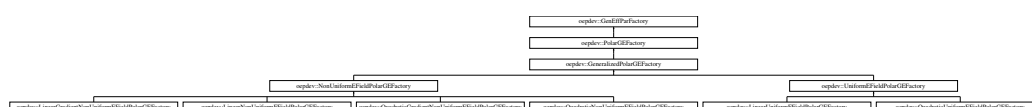
- oepdev/liboep/oep\_gdf.h
- oepdev/liboep/oep\_gdf.cc

## 18.72 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

### 18.72.1 Detailed Description

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,  $\mathbf{s} = (s^{[1]} \ s^{[2]} \ \dots)^T$ , 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

## 18.73 gefp.math.orthonorm.GrammSchmidt Class Reference

### Public Member Functions

- def **\_\_init\_\_** (self, V)
- def **normalize** (self)
- def **orthonormalize** (self)
- def **orthogonalize** (self)
- def **orthogonalize\_vector** (self, d, normalize=False)
- def **append** (self, d)
- def **proj** (self, u, v)

## Public Attributes

- **V**
- **n**

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

- gefp/gefp/math/orthonorm.py

## 18.74 oepdev::GramSchmidt Class Reference

Gram-Schmidt orthogonalization method.

```
#include <gram_schmidt.h>
```

## Public Member Functions

- [GramSchmidt](#) ()  
*Construct the blank Gram-Schmidt Orthonormalizer.*
- [GramSchmidt](#) (std::vector< psi::SharedVector > vectors)  
*Construct the Gram-Schmidt Orthonormalizer.*
- virtual [~GramSchmidt](#) ()  
*Destructor.*
- virtual std::vector< psi::SharedVector > [V](#) (void) const  
*Retrieve all the vectors.*
- virtual int [L](#) (void) const  
*Retrieve the number of vectors.*
- virtual psi::SharedVector [V](#) (int i) const  
*Retrieve the \*i\*th vector.*
- void [normalize](#) (void)  
*Normalize all the vectors.*
- void [orthonormalize](#) (void)  
*Orthonormalize all the vectors.*
- void [orthogonalize](#) (void)  
*Orthogonalize all the vectors.*
- void [orthogonalize\\_vector](#) (psi::SharedVector &d, bool [normalize](#)=false) const  
*Orthogonalize vector with respect to the vector set. Modifies **d**.*
- psi::SharedVector [projection](#) (psi::SharedVector u, psi::SharedVector v) const
- void [append](#) (psi::SharedVector d)  
*Append new vector to the list.*
- void [reset](#) (std::vector< psi::SharedVector > [V](#))

*Reset by providing new vectors.*

- void `reset` (void)

*Reset to empty state.*

## Protected Attributes

- `std::vector< psi::SharedVector > V_`

*Vectors stored.*

- `int L_`

*Number of vectors.*

### 18.74.1 Detailed Description

Orthonormalize a set of  $L$  vectors, i.e.,

$$\{\mathbf{v}_k\} \rightarrow \{\mathbf{u}_k\} \text{ for } k = 1, 2, \dots, L$$

## Implementation

The orthogonalized vectors are generated according to

$$\mathbf{u}_k = \left[ 1 - \sum_{i=1}^{k-1} \hat{P}_{\mathbf{u}_i} \right] \mathbf{v}_k$$

where the projection operator is given by

$$\hat{P}_{\mathbf{u}} = \frac{1}{u^2} \mathbf{u} [\square \cdot \mathbf{u}]$$

### 18.74.2 Constructor & Destructor Documentation

#### 18.74.2.1 GramSchmidt() [1/2]

```
oepdev::GramSchmidt::GramSchmidt ( )
```

#### 18.74.2.2 GramSchmidt() [2/2]

```
oepdev::GramSchmidt::GramSchmidt (
    std::vector< psi::SharedVector > vectors )
```



## Parameters

<code>vectors</code>	- list of vectors to be orthogonalized.
----------------------	---

## 18.74.3 Member Function Documentation

18.74.3.1 `projection()`

```
psi::SharedVector oepdev::GramSchmidt::projection (
    psi::SharedVector u,
    psi::SharedVector v ) const
```

Compute the projection vector.

## Parameters

<code>u</code>	- projected direction
<code>v</code>	- projected vector

## Returns

a new vector  $\mathbf{v}'$  such that

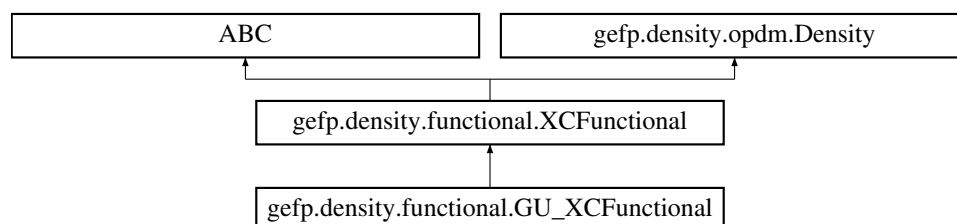
$$\mathbf{v}' = \hat{P}_{\mathbf{u}} \mathbf{v}$$

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

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

18.75 `gefp.density.functional.GU_XCFunctional` Class Reference

Inheritance diagram for `gefp.density.functional.GU_XCFunctional`:



## Public Member Functions

- `def __init__ (self)`
- `def abbr (self)`
- `def gradient_P_approximate_old (self, x)`
- `def gradient_P_approximate (self, x)`

## Static Public Member Functions

- `def name ()`
- `def fij (n)`
- `def fij_1 (n, m)`

## Additional Inherited Members

### 18.75.1 Detailed Description

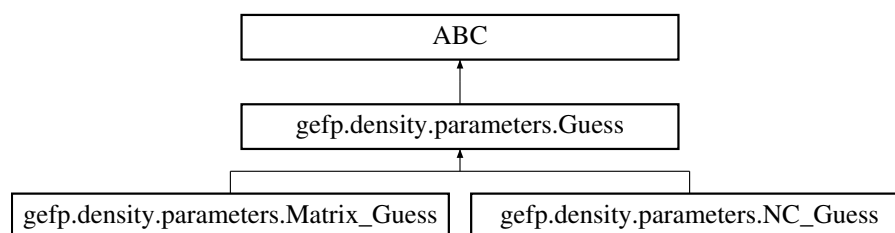
The Goedecker-Urmigar Exchange-Correlation Functional.

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

- `gefp/gefp/density/functional.py`

## 18.76 gefp.density.parameters.Guess Class Reference

Inheritance diagram for `gefp.density.parameters.Guess`:



## Public Member Functions

- `def __init__ (self, n=None, c=None, matrix=None)`
- `def create (cls, n=None, c=None, matrix=None, t='matrix')`
- `def update (self, S=None, C=None)`
- `def matrix (self)`
- `def copy (self)`
- `def pack (self)`

- def **unpack** (self)
- def **\_\_add\_\_** (self, other)
- def **\_\_sub\_\_** (self, other)
- def **\_\_rmul\_\_** (self, other)

### 18.76.1 Detailed Description

\

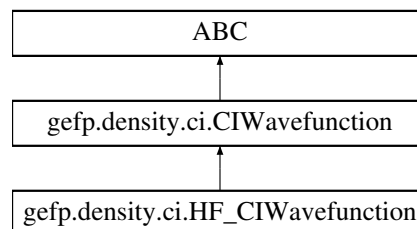
Container for handling density matrix guesses for DMFT calculations.  
Contains functionalities for working with occupation numbers,  
natural orbitals and density matrices.

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

- gefp/gefp/density/parameters.py

## 18.77 gefp.density.ci.HF\_CIWavefunction Class Reference

Inheritance diagram for gefp.density.ci.HF\_CIWavefunction:



### Public Member Functions

- def **\_\_init\_\_** (self, ref\_wfn, E)
- def **make\_ci\_l** (self)

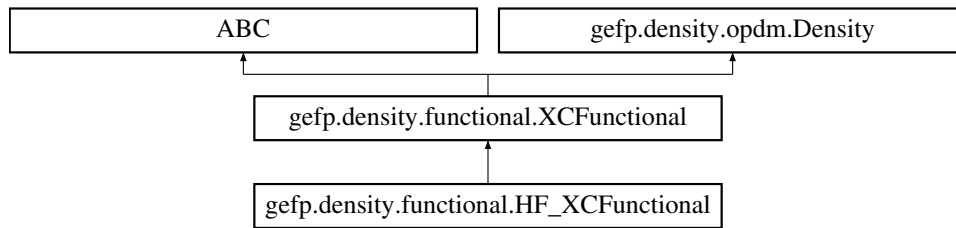
### Additional Inherited Members

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

- gefp/gefp/density/ci.py

## 18.78 gefp.density.functional.HF\_XCFunctional Class Reference

Inheritance diagram for gefp.density.functional.HF\_XCFunctional:



## Public Member Functions

- `def __init__ (self)`
- `def abbr (self)`
- `def energy_D (self, x, mode='scf-mo')`
- `def gradient_D (self, x)`
- `def gradient_nc (self, x)`

## Static Public Member Functions

- `def name ()`
- `def fij (n)`
- `def fij_1 (n, m)`

## Additional Inherited Members

### 18.78.1 Detailed Description

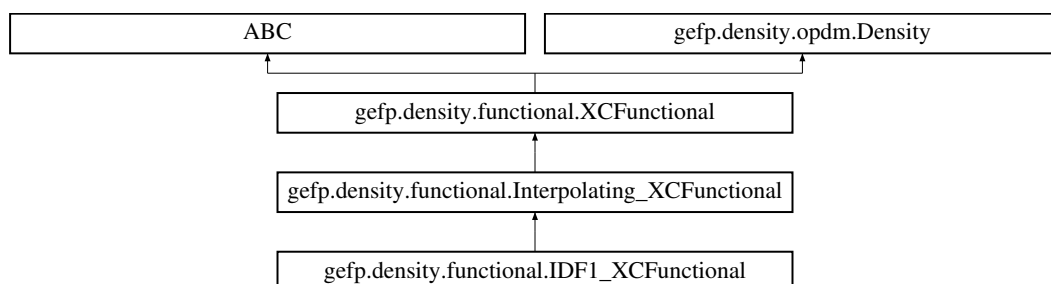
The Hartree-Fock Exchange-Correlation Functional

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

- `gefp/gefp/density/functional.py`

## 18.79 gefp.density.functional.IDF1\_XCFunctional Class Reference

Inheritance diagram for gefp.density.functional.IDF1\_XCFunctional:



## Public Member Functions

- def `__init__` (self, parameters)
- def `abbr` (self)

## Static Public Member Functions

- def `name` ()

## Additional Inherited Members

### 18.79.1 Detailed Description

4-Component Interpolation Functional with MBB lower and HF upper bound.

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

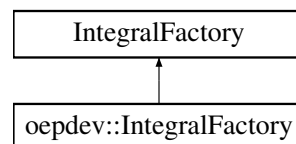
- `gefp/gefp/density/functional.py`

## 18.80 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.*

### 18.80.1 Detailed Description

In addition to integrals available in Psi4, `oepdev::IntegralFactory` enables to compute also:

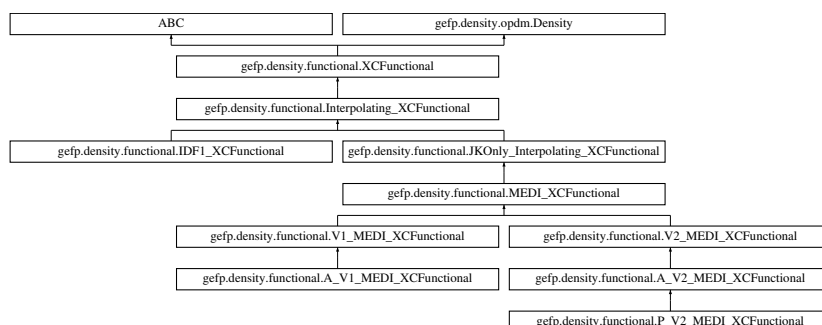
- OEI's:
  - none at that moment
- ERI's:
  - integrals of type (a|b) - `oepdev::ERI_1_1`
  - integrals of type (ab|c) - `oepdev::ERI_2_1`
  - integrals of type (abc|d) - `oepdev::ERI_3_1`
  - integrals of type (ab|cd) - `oepdev::ERI_2_2` (also in Psi4 as `psi::ERI`)

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

- `oepdev/libpsi/integral.h`
- `oepdev/libpsi/integral.cc`

## 18.81 `gefp.density.functional.Interpolating_XCFunctional` Class Reference

Inheritance diagram for `gefp.density.functional.Interpolating_XCFunctional`:



## Public Member Functions

- `def __init__ (self, parameters)`

## Static Public Member Functions

- `def fij (n, z, eps=1.0e-20)`

## Public Attributes

- `parameters`

## Additional Inherited Members

### 18.81.1 Detailed Description

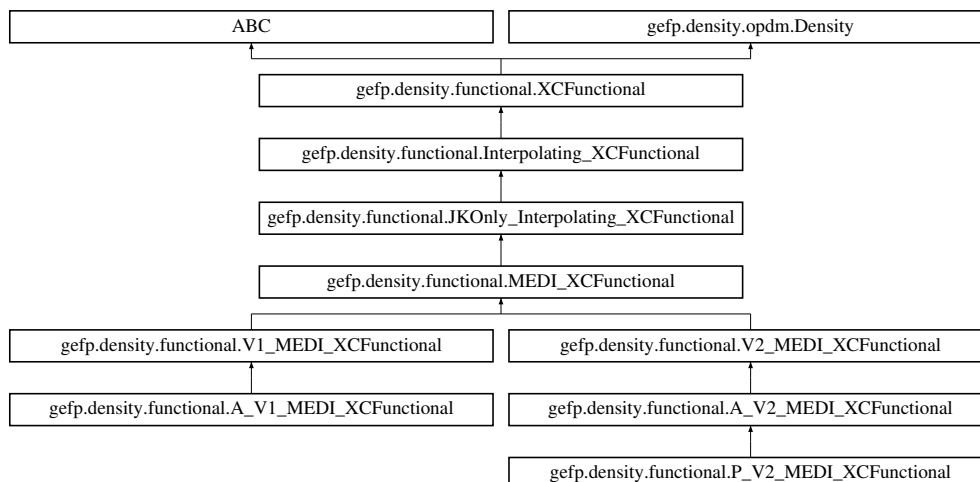
The New Class of Exchange-Correlation Functionals: Interpolation Functionals: Abstract

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

- `gefp/gefp/density/functional.py`

## 18.82 `gefp.density.functional.JKOnly_Interpolating_XCFunctional` Class Reference

Inheritance diagram for `gefp.density.functional.JKOnly_Interpolating_XCFunctional`:



## Public Member Functions

- `def __init__ (self, coeff, kmax=10)`
- `def compute_a0 (self, n)`
- `def compute_ak (self, k, t)`
- `def fij (self, n)`
- `def gradient_P (self, x)`

## Additional Inherited Members

### 18.82.1 Detailed Description

The New Class of Exchange-Correlation Functionals: Interpolation Functionals for JK-Onl

They differ in the model for the interpolation decay.  
Each functional has to provide its own coefficients,  
handled by 'coeff' dictionary in the constructor.

Eg.: `coeff = {'coefficient_name': coefficient_object}`

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

- `gefp/gefp/density/functional.py`

## 18.83 oepdev::KabschSuperimposer Class Reference

Compute the Cartesian rotation matrix between two structures.

```
#include <kabsch_superimposer.h>
```

## Public Member Functions

- [KabschSuperimposer \(\)](#)  
*Constructor.*
- [~KabschSuperimposer \(\)](#)  
*Destructor.*
- `void compute (psi::SharedMatrix initial_xyz, psi::SharedMatrix final_xyz)`  
*Run the Kabsch algorithm.*
- `void compute (psi::SharedMolecule initial_mol, psi::SharedMolecule final_mol)`  
*Run the Kabsch algorithm.*
- `psi::SharedMatrix get_transformed (void)`  
*Return transformed coordinates  $\mathbf{X}'$ .*
- `double rms (void)`  
*Compute RMS or superimposition.*



- void `clear` (void)  
*Clear all previous calculations.*

## Public Attributes

- psi::SharedMatrix `rotation`  
*Rotation matrix  $\mathbf{r}$ .*
- psi::SharedVector `translation`  
*Translation vector  $\mathbf{t}$ .*
- psi::SharedMatrix `initial_xyz`  
*Initial xyz  $\mathbf{X}$ .*
- psi::SharedMatrix `final_xyz`  
*Final xyz  $\mathbf{X}_0$ .*

### 18.83.1 Detailed Description

The superimposition is defined as:

$$\mathbf{X}' = \mathbf{t} + \mathbf{X} \cdot \mathbf{r} \approx \mathbf{X}_0$$

where  $X_{iu}$  is the  $u$ -th Cartesian component of the  $i$ -th atom's position,  $\mathbf{t}$  is the superimposition translation vector,  $\mathbf{r}$  is the superimposition rotation matrix, and prime denotes transformed coordinates.

The superimposition uses the Kabsch algorithm.

#### The Kabsch Algorithm.

Rotation matrix is calculated from

$$\mathbf{r} = \mathbf{U} \cdot \mathbf{V}^T$$

where

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T$$

is the singular value decomposition of the covariance matrix

$$\mathbf{A} = [\mathbf{X} - \langle \mathbf{X} \rangle]^T \cdot [\mathbf{X}_0 - \langle \mathbf{X}_0 \rangle]$$

The average of position is given by

$$\langle \mathbf{X} \rangle_u = \frac{1}{N} \sum_i X_{iu}$$

where  $N$  is the number of atoms. If determinant of rotation matrix is negative (indicating inversion), rotation matrix is recomputed by inverting the sign of the third column of  $\mathbf{V}$ .

The translation vector is then calculated by

$$\mathbf{t} = \langle \mathbf{X}_0 \rangle - \langle \mathbf{X} \rangle \cdot \mathbf{r}$$

## 18.83.2 Member Function Documentation

### 18.83.2.1 compute() [1/2]

```
void oepdev::KabschSuperimposer::compute (
    psi::SharedMatrix initial_xyz,
    psi::SharedMatrix final_xyz )
```

#### Parameters

<i>initial_xyz</i>	- position vectors <b>X</b>
<i>final_xyz</i>	- position vectors <b>X</b> <sub>0</sub>

### 18.83.2.2 compute() [2/2]

```
void oepdev::KabschSuperimposer::compute (
    psi::SharedMolecule initial_mol,
    psi::SharedMolecule final_mol ) [inline]
```

#### Parameters

<i>initial_mol</i>	- molecule with atomic positions at <b>X</b>
<i>final_mol</i>	- molecule with atomic positions at <b>X</b> <sub>0</sub>

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

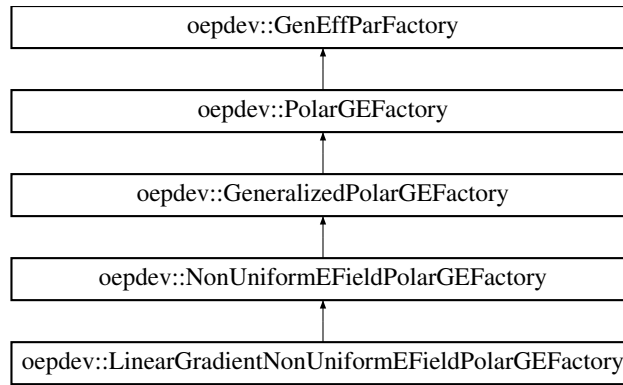
- oepdev/libutil/[kabsch\\_superimposer.h](#)
- oepdev/libutil/kabsch\_superimposer.cc

## 18.84 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

### 18.84.1 Detailed Description

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:

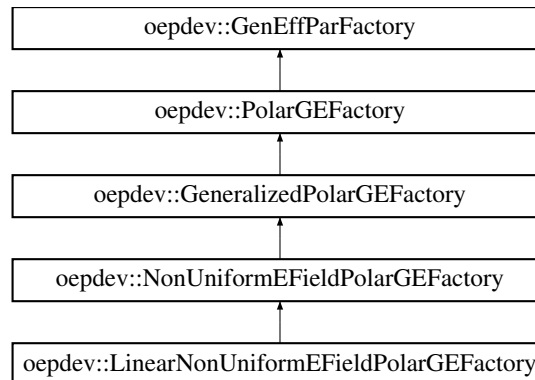
- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp\_polar\_nonuniform\_field\_1\_grad\_1.cc

## 18.85 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

#### 18.85.1 Detailed Description

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:

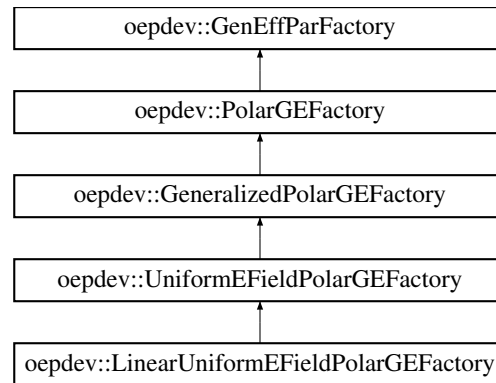
- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp\_polar\_nonuniform\_field\_1.cc

## 18.86 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

#### 18.86.1 Detailed Description

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

## 18.87 `gefp.density.population.Loc` Class Reference

### Public Member Functions

- `def __init__ (self, wfn, method='BOYS')`
- `def lmoc (self)`
- `def el_charges (self)`
- `def el_dipoles (self)`
- `def el_quadrupoles (self)`
- `def __repr__ (self)`

### Public Attributes

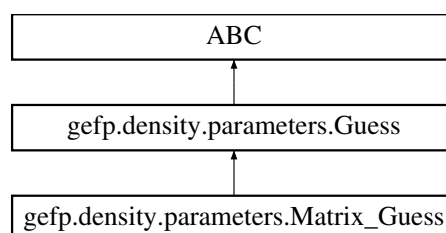
- **wfn**
- **method**
- **La**
- **Lb**
- **Ua**
- **Ub**

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

- `gefp/gefp/density/population.py`

## 18.88 `gefp.density.parameters.Matrix_Guess` Class Reference

Inheritance diagram for `gefp.density.parameters.Matrix_Guess`:



### Public Member Functions

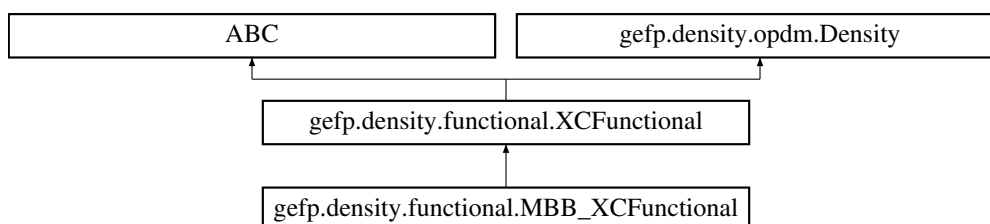
- `def __init__ (self, n=None, c=None, matrix=None)`
- `def pack (self)`

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

- `gefp/gefp/density/parameters.py`

## 18.89 gefp.density.functional.MBB\_XCFunctional Class Reference

Inheritance diagram for gefp.density.functional.MBB\_XCFunctional:



### Public Member Functions

- `def __init__ (self)`
- `def abbr (self)`
- `def energy_P (self, x)`
- `def energy_pc (self, x)`
- `def gradient_P (self, x)`
- `def gradient_pc (self, x)`
- `def energy_D (self, x, mode='scf-mo')`

### Static Public Member Functions

- `def name ()`
- `def fij (n)`

### Additional Inherited Members

#### 18.89.1 Detailed Description

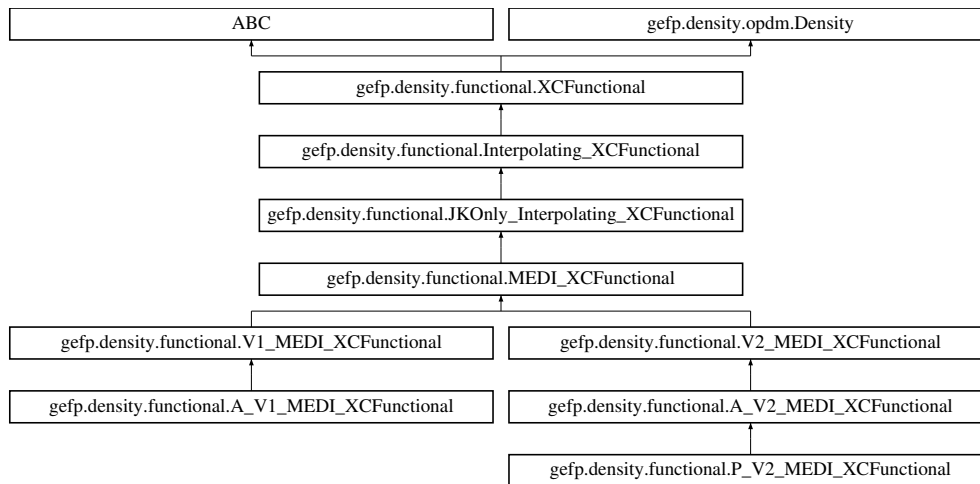
The Muller-Buijse-Baerends Exchange-Correlation Functional.

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

- `gefp/gefp/density/functional.py`

## 18.90 gefp.density.functional.MEDI\_XCFunctional Class Reference

Inheritance diagram for gefp.density.functional.MEDI\_XCFunctional:



## Public Member Functions

- `def __init__ (self, coeff, kmax)`
- `def compute_t (self, n=None, c=None)`
- `def fij (self, n)`

## Additional Inherited Members

### 18.90.1 Detailed Description

The New Class of Exchange-Correlation Functionals:  
Interpolation Functionals with Monotonous Exponential Decay.

The decay in the interpolates is modelled by the monotonous decay

$$a_k = a_0 \exp(k \ln\{1 - a_0\})$$

Functional coefficients:

- `'a0'` - first term in the interpolates coefficient  
(the one that multiplies MBB functional term - for  $k=0$ ).  
Parameter `'a0'` has to be between 0.0 and 1.0. The smaller `'a0'`,  
the more terms need to be taken (larger  $k_{\max}$ ).

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

- `gefp/gefp/density/functional.py`

## 18.91 oepdev::MultipoleConvergence Class Reference

Multipole Convergence.

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



## Public Types

- enum [ConvergenceLevel](#) {  
    **R1, R2, R3, R4,**  
    **R5** }
- enum [Property](#) { **Energy, Potential, Field** }

## Public Member Functions

- [MultipoleConvergence](#) (std::shared\_ptr< [DMTPole](#) > dmt1, std::shared\_ptr< [DMTPole](#) > dmt2, [ConvergenceLevel](#) max\_clevel=R5)  
*Construct from two shared [DMTPole](#) objects.*
- virtual [~MultipoleConvergence](#) ()  
*Destructor.*
- void [compute](#) ([Property](#) property=Energy)
- void [compute](#) (const double &x, const double &y, const double &z, [Property](#) property=Potential)
- std::shared\_ptr< psi::Matrix > [level](#) ([ConvergenceLevel](#) clevel=R5)

## Protected Member Functions

- void [compute\\_energy](#) ()  
*Compute the generalized energy.*
- void [compute\\_potential](#) (const double &x, const double &y, const double &z)  
*Compute the generalized potential.*
- void [compute\\_field](#) (const double &x, const double &y, const double &z)  
*Compute the generalized field potential.*

## Protected Attributes

- [ConvergenceLevel max\\_clevel\\_](#)  
*Maximum allowed convergence level.*
- std::shared\_ptr< [DMTPole](#) > [dmt1\\_](#)  
*First DMTP set.*
- std::shared\_ptr< [DMTPole](#) > [dmt2\\_](#)  
*Second DMTP set.*
- std::map< std::string, std::shared\_ptr< psi::Matrix > > [convergenceList\\_](#)  
*Dictionary of available convergence level results.*
- std::map< std::string, std::shared\_ptr< psi::Matrix > > [energyConvergencePairs\\_](#)  
*Dictionary of available energy convergence pairs.*

### 18.91.1 Detailed Description

Handles the convergence of the distributed multipole expansions up to hexadecapole. Takes shared pointers to existing [DMTPole](#) objects and computes the generalized property:

- energy
- potential from the DMTP sets. The results are stored in matrix of size (N1, N2) where N1 and N2 are equal to the number of DMTP's in a set described by according [DMTPole](#) object given.

#### Note

Useful options:

- DMTP\_CONVER - level of multipole series convergence (available: R1, R2, R3, R4 and R5). Default: R5.

#### See also

[DMTPole](#)

### 18.91.2 Member Enumeration Documentation

#### 18.91.2.1 ConvergenceLevel

```
enum oepdev::MultipoleConvergence::ConvergenceLevel
```

Convergence level of the multipole expansion:

##### Parameters

<i>R1</i>	- qq term
<i>R2</i>	- qd and sum of the above
<i>R3</i>	- qQ, dd and sum of the above
<i>R4</i>	- qO, dQ and sum of the above
<i>R5</i>	- qH, dO, QQ and sum of the above

#### 18.91.2.2 Property

```
enum oepdev::MultipoleConvergence::Property
```

Property to be evaluated from DMTP's:

## Parameters

<i>Energy</i>	- generalized energy
<i>Field</i>	- generalized field
<i>Potential</i>	- generalized potential

## 18.91.3 Constructor &amp; Destructor Documentation

## 18.91.3.1 MultipoleConvergence()

```
oepdev::MultipoleConvergence::MultipoleConvergence (
    std::shared_ptr< DMTPole > dmt1,
    std::shared_ptr< DMTPole > dmt2,
    MultipoleConvergence::ConvergenceLevel max_clevel = R5 )
```

## Parameters

<i>dmt1</i>	- first <a href="#">DMTPole</a> object
<i>dmt2</i>	- second <a href="#">DMTPole</a> object
<i>max_clevel</i>	- maximul allowed convergence level

## 18.91.4 Member Function Documentation

## 18.91.4.1 compute() [1/2]

```
void oepdev::MultipoleConvergence::compute (
    MultipoleConvergence::Property property = Energy )
```

Compute the generalized interaction property

## Parameters

<i>property</i>	- generalized Property
-----------------	------------------------

## 18.91.4.2 compute() [2/2]

```
void oepdev::MultipoleConvergence::compute (
```

```

const double & x,
const double & y,
const double & z,
MultipoleConvergence::Property property = Potential )

```

Compute the generalized generator property

#### Parameters

<i>x</i>	- location x-th Cartesian component
<i>y</i>	- location y-th Cartesian component
<i>z</i>	- location z-th Cartesian component
<i>property</i>	- generalized Property

#### 18.91.4.3 level()

```

std::shared_ptr< psi::Matrix > oepdev::MultipoleConvergence::level (
    MultipoleConvergence::ConvergenceLevel clevel = R5 )

```

Grab the generalized property at specified level of convergence

#### Parameters

<i>clevel</i>	- ConvergenceLevel
---------------	--------------------

#### Returns

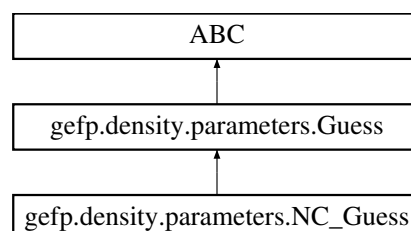
vector of results (each element corresponds to each DMTP pair in a set)

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

- oepdev/lib3d/dmtp.h
- oepdev/lib3d/dmtp\_base.cc

## 18.92 gefp.density.parameters.NC\_Guess Class Reference

Inheritance diagram for gefp.density.parameters.NC\_Guess:



## Public Member Functions

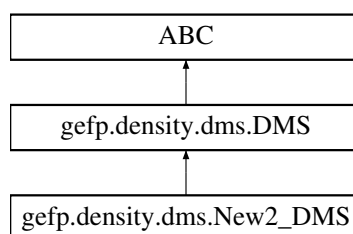
- `def __init__ (self, n=None, c=None, matrix=None)`
- `def pack (self)`

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

- `gefp/gefp/density/parameters.py`

## 18.93 gefp.density.dms.New2\_DMS Class Reference

Inheritance diagram for gefp.density.dms.New2\_DMS:



## Public Member Functions

- `def __init__ (self, type='da')`

### 18.93.1 Detailed Description

The order of DMS blocks:

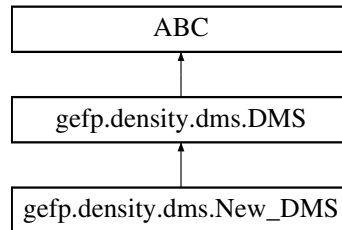
Block	DMS order	Interaction	Symmetry	Dimension	Group
-----	-----	-----	-----	-----	-----
1.	(1, 0)	Induction	Symmetric	(n, n, N, 3)	Z (1)
2.	(2, 0)	Induction	Symmetric	(n, n, N, 3, 3)	Z (1)
3.	(0, 1)	Pauli	Asymmteric	(n, n)	Z (1)
4.	(0, 3)	Pauli	Asymmetric	(n, n)	Z (1)
5.	(0, 5)	Pauli	Asymmetric	(n, n)	Z (1)
6.	(0, 2)	Pauli	Asymmetric	(n, n)	Z (2)
7.	(0, 4)	Pauli	Asymmetric	(n, n)	Z (2)
8.	(0, 6)	Pauli	Asymmetric	(n, n)	Z (2)

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

- `gefp/gefp/density/dms.py`

## 18.94 gefp.density.dms.New\_DMS Class Reference

Inheritance diagram for gefp.density.dms.New\_DMS:



### Public Member Functions

- `def __init__ (self, type='da')`

### 18.94.1 Detailed Description

The order of DMS blocks:

Block	DMS order	Interaction	Symmetry	Dimension	Group
1.	(1, 0)	Induction	Symmetric	(n, n, N, 3)	Z (1)
2.	(2, 0)	Induction	Symmetric	(n, n, N, 3, 3)	Z (1)
3.	(0, 1)	Pauli	Asymmteric	(n, n)	Z (1)
4.	(0, 3)	Pauli	Asymmetric	(n, n)	Z (1)
5.	(0, 2)	Pauli	Asymmetric	(n, n)	Z (2)
6.	(0, 4)	Pauli	Asymmetric	(n, n)	Z (2)

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

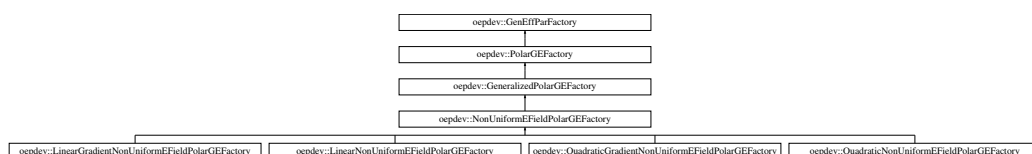
- `gefp/gefp/density/dms.py`

## 18.95 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

### 18.95.1 Detailed Description

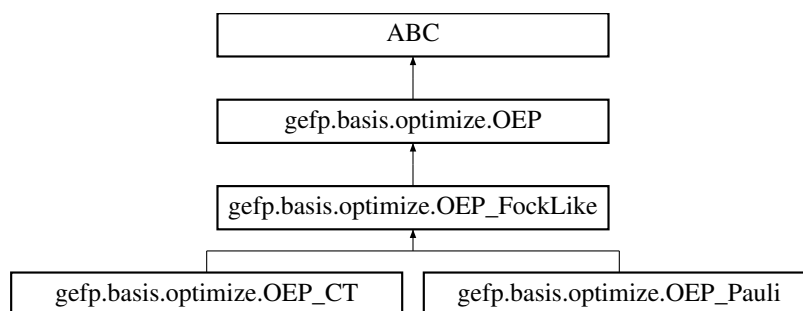
Implements a class of density matrix susceptibility models for parameterization in the non-uniform electric field generated by point charges.

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

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

## 18.96 gefp.basis.optimize.OEP Class Reference

Inheritance diagram for gefp.basis.optimize.OEP:



## Public Member Functions

- def **\_\_init\_\_** (self, wfn, dfbasis)
- def **create** (cls, name, wfn, dfbasis)

## Public Attributes

- **wfn**
- **dfbasis**
- **mints**
- **basis\_test**
- **basis\_prim**
- **basis\_aux**
- **eri\_pppt**
- **V**

### 18.96.1 Detailed Description

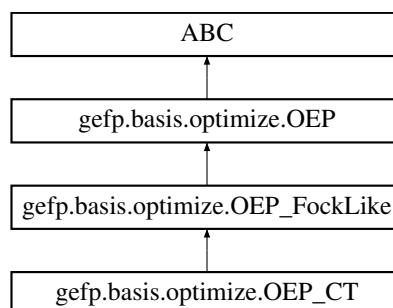
OEP object that defines the V matrix necessary for GDF.

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

- `gefp/gefp/basis/optimize.py`

## 18.97 gefp.basis.optimize.OEP\_CT Class Reference

Inheritance diagram for `gefp.basis.optimize.OEP_CT`:



## Public Member Functions

- `def __init__(self, wfn, dfbasis)`

## Additional Inherited Members

### 18.97.1 Detailed Description

OEP for Group-(i) term of Otto-Ladik's theory of Charge-Transfer Energy

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

- `gefp/gefp/basis/optimize.py`

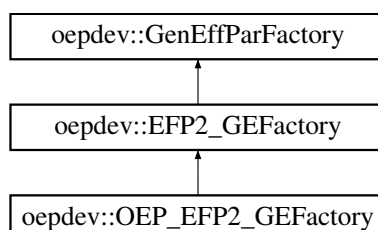


## 18.98 oepdev::OEP\_EFP2\_GEFactory Class Reference

OEP-EFP2 GEFP Factory.

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

Inheritance diagram for oepdev::OEP\_EFP2\_GEFactory:



### Public Member Functions

- [OEP\\_EFP2\\_GEFactory](#) (std::shared\_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)  
*Construct from Psi4 options.*
- virtual [~OEP\\_EFP2\\_GEFactory](#) ()  
*Destruct.*
- virtual std::shared\_ptr< [GenEffPar](#) > [compute](#) (void)  
*Compute the OEP-EFP2 parameters.*

### Additional Inherited Members

#### 18.98.1 Detailed Description

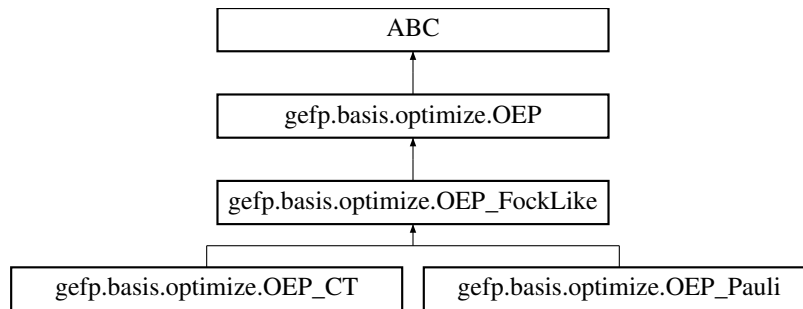
Basic interface for the OEP-EFP2 parameters.

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

- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp\_oep\_efp2.cc

## 18.99 gefp.basis.optimize.OEP\_FockLike Class Reference

Inheritance diagram for gefp.basis.optimize.OEP\_FockLike:



## Public Member Functions

- `def __init__ (self, wfn, dfbasis)`

## Additional Inherited Members

### 18.99.1 Detailed Description

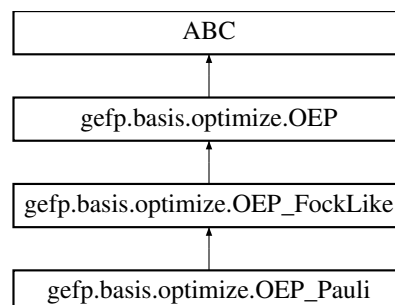
OEP for S1 term in Murrell et al.'s theory of Pauli repulsion

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

- `gefp/gefp/basis/optimize.py`

## 18.100 gefp.basis.optimize.OEP\_Pauli Class Reference

Inheritance diagram for gefp.basis.optimize.OEP\_Pauli:



## Public Member Functions

- `def __init__ (self, wfn, dfbasis)`

## Additional Inherited Members

### 18.100.1 Detailed Description

OEP for S1 term in Murrell et al.'s theory of Pauli repulsion

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

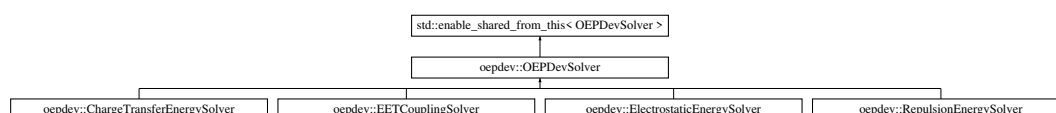
- `gefp/gefp/basis/optimize.py`

## 18.101 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\\_oep\\_based](#) (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.*
- psi::Options & [options\\_](#)  
*Options.*
- std::vector< std::string > [methods\\_oepBased\\_](#)  
*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.*

### 18.101.1 Detailed Description

Uses only a wavefunction union object to initialize.

## Available solvers

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

## Options

### Interaction Property Method

- OEPDEV\_SOLVER\_EINT\_COUL\_AO - Coulombic energy: AO expanded
- OEPDEV\_SOLVER\_EINT\_COUL\_MO - Coulombic energy: MO expanded
- OEPDEV\_SOLVER\_EINT\_COUL\_ESP - Coulombic energy: ESP
- OEPDEV\_SOLVER\_EINT\_COUL\_CAMM - Coulombic energy: [CAMM](#)
- OEPDEV\_SOLVER\_EINT\_REP\_HS - Exchange-repulsion energy: Hayes-Stone
- OEPDEV\_SOLVER\_EINT\_REP\_DDS - Exchange-repulsion energy: DDS
- OEPDEV\_SOLVER\_EINT\_REP\_MRW - Exchange-repulsion energy: Murrell et al.
- OEPDEV\_SOLVER\_EINT\_REP\_OL - Exchange-repulsion energy: Otto-Ladik
- OEPDEV\_SOLVER\_EINT\_REP\_OEP1 - Exchange-repulsion energy: OEP (S1: GDF, S2: ESP)

- OEPDEV\_SOLVER\_EINT\_REP\_OEP2 - Exchange-repulsion energy: OEP (S1: GDF, S2: CAMM)
- OEPDEV\_SOLVER\_EINT\_REP\_EFP2 - Exchange-repulsion energy: EFP2
- OEPDEV\_SOLVER\_EINT\_CT\_OL - Charge-transfer energy: Otto-Ladik
- OEPDEV\_SOLVER\_EINT\_CT\_OEP - Charge-transfer energy: OEP
- OEPDEV\_SOLVER\_EINT\_CT\_EFP2 - Charge-transfer energy: EFP2

#### Generalized density fitting (GDF) options:

- OEPDEV\_DF\_TYPE - type of the GDF. Default: DOUBLE. Other: SINGLE.
- 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.

#### EFP2 Charge transfer energy options:

- EFP2\_CT\_POTENTIAL\_INTS - Type of potential one-electron operator. Default: 'DMTP'. Other: 'ERI'.
- EFP2\_CT\_NO\_OCTUPOLES - Ignore octupole moments from potential integrals? Default: True.

#### Excited States

- EXCITED\_STATE - ID of state for all monomers to consider. If  $-n$ , then the  $n$ th bright state is taken. Default:  $-1$ .
- EXCITED\_STATE\_A - ID of state for monomer A to consider. If  $-n$ , then the  $n$ th bright state is taken. Default:  $-1$ .
- EXCITED\_STATE\_B - ID of state for monomer B to consider. If  $-n$ , then the  $n$ th bright state is taken. Default:  $-1$ .
- OSCILLATOR\_STRENGTH\_THRESHOLD - Threshold for oscillator strength for bright states selection. Default:  $0.01$ .
- TrCAMM\_SYMMETRIZE - Whether to use the 'symmetrized transition density' or not. Default: true.
- TI\_CIS\_SCF\_FOCK\_MATRIX - Whether to compute the full SCF Fock matrix for the dimer or approximate it from monomer OPDM's. Default: false.
- TI\_CIS\_PRINT\_FOCK\_MATRIX - Whether to print the Fock matrix (AB block in AO basis) or not. Default: false.

## Environmental variables

One can easily access those variables from Python level by calling

```
psi4.get_variable("name of variable")
```

in your Python script.

Table 18.84: Environmental variables in the OEPDev solver.

Keyword	Description
<b>Coulombic Interaction Energy</b>	
<i>Distributed Multipole Series</i>	
EINT COUL CAMM R-1	CAMM charge-charge terms
EINT COUL CAMM R-2	CAMM charge-dipole terms + all above
EINT COUL CAMM R-3	CAMM charge-quadrupole, dipole-dipole + all above
EINT COUL CAMM R-4	CAMM charge-octupole, dipole-quadrupole + all above
EINT COUL CAMM R-5	CAMM charge-hexadecapole, dipole-octupole, quadrupole-quadrupole + all above
EINT COUL ESP	ESP charge-charge terms
<i>Exact First-Order Perturbation Theory</i>	
EINT COUL EXACT	MO or AO expanded Coulombic energy. Both give same results but MO is much faster.
<b>Exchange-Repulsion Interaction Energy</b>	
<i>Density Decomposition Scheme</i>	
EINT REP DDS KCAL	Pauli repulsion
EINT EXC DDS KCAL	DDS exchange
EINT EXR DDS KCAL	Sum of the above
<i>Hayes-Stone model</i>	
EINT REP HAYES-STONE KCAL	Pauli repulsion
EINT EXC HAYES-STONE KCAL	Pure exchange
EINT EXR HAYES-STONE KCAL	Sum of the above

Keyword	Description
<b><i>Murrell et al. model</i></b>	
EINT REP MURRELL-ETAL KCAL	Pauli repulsion
EINT EXC MURRELL-ETAL KCAL	Pure exchange (same as Hayes-Stone)
EINT EXR MURRELL-ETAL KCAL	Sum of the above
EINT REP MURRELL-ETAL:S1 KCAL	Pauli repulsion: $S^{-1}$ term
EINT REP MURRELL-ETAL:S2 KCAL	Pauli repulsion: $S^{-2}$ term
<b><i>Otto-Ladik model</i></b>	
EINT REP OTTO-LADIK KCAL	Pauli repulsion
EINT EXC OTTO-LADIK KCAL	Pure exchange (same as Hayes-Stone)
EINT EXR OTTO-LADIK KCAL	Sum of the above
EINT REP OTTO-LADIK:S1 KCAL	Pauli repulsion: $S^{-1}$ term
EINT REP OTTO-LADIK:S2 KCAL	Pauli repulsion: $S^{-2}$ term
<b><i>EFP2 model</i></b>	
EINT REP EFP2 KCAL	Pauli repulsion
EINT EXC EFP2 KCAL	Exchange: SGO approximation of Jensen
EINT EXR EFP2 KCAL	Sum of the above
EINT REP EFP2:S1 KCAL	Pauli repulsion: $S^{-1}$ term
EINT REP EFP2:S2 KCAL	Pauli repulsion: $S^{-2}$ term
<b><i>OEP-based models</i></b>	
EINT REP OEP-MURRELL-ETAL-1 KCAL	Pauli repulsion: S1 term using GDF, S2 term using <a href="#">CAMM</a>
EINT REP OEP-MURRELL-ETAL-1 S1 KCAL	$S^{-1}$ term of the above total term
EINT REP OEP-MURRELL-ETAL-1 S2 KCAL	$S^{-2}$ term of the above total term
EINT REP OEP-MURRELL-ETAL-2 KCAL	Pauli repulsion: S1 term using GDF, S2 term using ESP
EINT REP OEP-MURRELL-ETAL-2 S1 KCAL	$S^{-1}$ term of the above total term
EINT REP OEP-MURRELL-ETAL-2 S2 KCAL	$S^{-2}$ term of the above total term
<b>Charge-Transfer Interaction Energy</b>	

Keyword	Description
<b><i>EFP2 Model</i></b>	
EINT CT EFP2 KCAL	Total charge-transfer energy (kcal/mole)
<b><i>Otto-Ladik Model</i></b>	
EINT CT OTTO-LADIK KCAL	Total charge-transfer energy (kcal/mole)
<b><i>OEP-Based Otto-Ladik Model</i></b>	
EINT CT OEP-OTTO-LADIK KCAL	Total charge-transfer energy (kcal/mole)
<b>EET Coupling Constant</b>	
<b><i>TrCamm Model</i></b>	
EET V0 TRCamm R1 CM-1	Overlap-uncorrected, converged to R1 (cm-1)
EET V TRCamm R1 CM-1	Overlap-corrected, converged to R1 (cm-1)
EET V0 TRCamm R2 CM-1	Overlap-uncorrected, converged to R2 (cm-1)
EET V TRCamm R2 CM-1	Overlap-corrected, converged to R2 (cm-1)
EET V0 TRCamm R3 CM-1	Overlap-uncorrected, converged to R3 (cm-1)
EET V TRCamm R3 CM-1	Overlap-corrected, converged to R3 (cm-1)
EET V0 TRCamm R4 CM-1	Overlap-uncorrected, converged to R4 (cm-1)
EET V TRCamm R4 CM-1	Overlap-corrected, converged to R4 (cm-1)
EET V0 TRCamm R5 CM-1	Overlap-uncorrected, converged to R5 (cm-1)
EET V TRCamm R5 CM-1	Overlap-corrected, converged to R5 (cm-1)
<b><i>TI/CIS Model</i></b>	
EET V0 COUL CM-1	Overlap-uncorrected Coulomb (Forster) coupling (cm-1)
EET V0 EXCH CM-1	Overlap-uncorrected exchange (Dexter) coupling (cm-1)
EET V COUL CM-1	Overlap-corrected Coulomb (Forster) coupling (cm-1)
EET V EXCH CM-1	Overlap-corrected exchange (Dexter) coupling (cm-1)
EET V OVRL CM-1	Remaining overlap correction to direct coupling(cm-1)
EET V0 ET1 CM-1	Overlap-uncorrected H <sub>13</sub> matrix element (cm-1)



Keyword	Description
EET V0 ET2 CM-1	Overlap-uncorrected H <sub>24</sub> matrix element (cm-1)
EET V0 HT1 CM-1	Overlap-uncorrected H <sub>14</sub> matrix element (cm-1)
EET V0 HT2 CM-1	Overlap-uncorrected H <sub>23</sub> matrix element (cm-1)
EET V0 CT CM-1	Overlap-uncorrected H <sub>34</sub> matrix element (cm-1)
EET V ET1 CM-1	Overlap-corrected H <sub>13</sub> matrix element (cm-1)
EET V ET2 CM-1	Overlap-corrected H <sub>24</sub> matrix element (cm-1)
EET V HT1 CM-1	Overlap-corrected H <sub>14</sub> matrix element (cm-1)
EET V HT2 CM-1	Overlap-corrected H <sub>23</sub> matrix element (cm-1)
EET V CT CM-1	Overlap-corrected H <sub>34</sub> matrix element (cm-1)
EET V0 TI-2 CM-1	Approximate 2nd-order indirect coupling (cm-1)
EET V0 TI-3 CM-1	Approximate 3rd-order indirect coupling (cm-1)
EET V TI-2 CM-1	2nd-order indirect coupling (cm-1)
EET V TI-3 CM-1	3rd-order indirect coupling (cm-1)
EET V0 DIRECT CM-1	Approximate direct coupling (cm-1)
EET V0 INDIRECT CM-1	Approximate indirect coupling (cm-1)
EET V DIRECT CM-1	Direct coupling (cm-1)
EET V INDIRECT CM-1	Indirect coupling (cm-1)
EET V0 TI-CIS CM-1	Approximate total coupling (cm-1)
EET V TI-CIS CM-1	Total coupling (cm-1)
EET V0 EXCH-M CM-1	Overlap-uncorrected exchange (Dexter) coupling in Mulliken approximation (cm-1)
EET V EXCH-M CM-1	Overlap-corrected exchange (Dexter) coupling in Mulliken approximation (cm-1)
EET V0 CT-M CM-1	Overlap-uncorrected H <sub>34</sub> matrix element in Mulliken approximation (cm-1)
EET V CT-M CM-1	Overlap-corrected H <sub>34</sub> matrix element in Mulliken approximation (cm-1)
<b><i>OEP-Based TI/CIS Model</i></b>	
EET V OEP:COUL CM-1	Overlap-corrected Coulomb (Forster) coupling (TrCamm; cm-1)

Keyword	Description
EET V OEP:EXCH CM-1	Overlap-corrected exchange (Dexter) coupling (Mulliken approximation of AO ERI's; cm-1)
EET V OEP:OVRL CM-1	Remaining overlap correction to direct coupling (cm-1)
EET V0 OEP:ET1 CM-1	Overlap-uncorrected H_13 matrix element (cm-1)
EET V0 OEP:ET2 CM-1	Overlap-uncorrected H_24 matrix element (cm-1)
EET V0 OEP:HT1 CM-1	Overlap-uncorrected H_14 matrix element (cm-1)
EET V0 OEP:HT2 CM-1	Overlap-uncorrected H_23 matrix element (cm-1)
EET V0 OEP:CT:CAMM CM-1	Overlap-uncorrected H_34 matrix element: CAMM approximation of ionic interaction (cm-1)
EET V0 OEP:CT:CC CM-1	Overlap-uncorrected H_34 matrix element: Point-charge approximation of ionic interaction (cm-1)
EET V OEP:ET1 CM-1	Overlap-corrected H_13 matrix element (cm-1)
EET V OEP:ET2 CM-1	Overlap-corrected H_24 matrix element (cm-1)
EET V OEP:HT1 CM-1	Overlap-corrected H_14 matrix element (cm-1)
EET V OEP:HT2 CM-1	Overlap-corrected H_23 matrix element (cm-1)
EET V OEP:CT:CAMM CM-1	Overlap-corrected H_34 matrix element: CAMM approximation of ionic interaction (cm-1)
EET V OEP:CT:CC CM-1	Overlap-corrected H_34 matrix element: Point-charge approximation of ionic interaction (cm-1)
EET V OEP:TI-2 CM-1	2nd-order indirect coupling (cm-1)
EET V OEP:TI-3:CAMM CM-1	3rd-order indirect coupling with CAMM approximation for V_CT (cm-1)
EET V OEP:TI-3:CC CM-1	3rd-order indirect coupling with point-charge approximation for V_CT (cm-1)
EET V OEP:DIRECT CM-1	Direct coupling (cm-1)
EET V OEP:INDIRECT:CAMM CM-1	Indirect coupling with CAMM approximation for V_CT (cm-1)
EET V OEP:INDIRECT:CC CM-1	Indirect coupling with point-charge approximation for V_CT (cm-1)
EET V OEP:TI-CIS:CAMM CM-1	Total coupling with CAMM approximation for V_CT (cm-1)

Keyword	Description
EET V OEP:TI-CIS:CC CM-1	Total coupling with point-charge approximation for V_CT (cm-1)

## 18.101.2 Constructor & Destructor Documentation

### 18.101.2.1 OEPDevSolver()

```
OEPDevSolver::OEPDevSolver (
    SharedWavefunctionUnion wfn_union )
```

#### Parameters

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

## 18.101.3 Member Function Documentation

### 18.101.3.1 build()

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

#### 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](#)

### 18.101.3.2 compute\_benchmark()

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

Each solver object has one `DEFAULT` benchmark method

#### Parameters

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

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

### 18.101.3.3 compute\_oep\_based()

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

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

#### Parameters

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

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

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

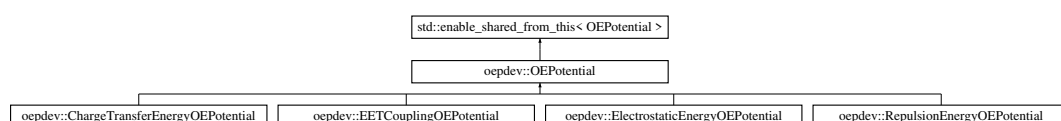
- [oepdev/libsolver/solver.h](#)
- [oepdev/libsolver/solver\\_base.cc](#)

## 18.102 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 [localize](#) (void)  
*Localize Occupied MO's.*
- virtual std::vector< psi::SharedVector > [mo\\_centroids](#) (psi::SharedMatrix C)  
*Compute MO centroids from LCAO-MO matrix.*
- 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.*
- [OEType](#) [oep](#) (const std::string &oepType) const  
*Retrieve the potentials.*
- SharedMatrix [matrix](#) (const std::string &oepType) const  
*Retrieve the potentials of a particular OEP type in a matrix form.*
- int [n](#) (const std::string &oepType) const  
*Retrieve the number of a particular OEP type.*
- SharedWavefunction [wfn](#) () const

*Retrieve wavefunction object.*

- SharedMatrix **cOcc** () const
- SharedMatrix **cVir** () const
- SharedMatrix **IOcc** () const
- SharedLocalizer **localizer** () const

*Retrieve MO Localizer.*

- std::vector< std::shared\_ptr< psi::Vector > > **lmoc** () const

*Retrieve LMO Centroids.*

- void **set\_name** (const std::string &**name**)

*Set the name of this OEP.*

- virtual void **print\_header** () const =0

*Header information.*

- void **print** () const

*Print the contents (OEP data)*

## 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\_**

*Promary Basis set.*

- SharedBasisSet **auxiliary\_**

*Auxiliary Basis set.*

- SharedBasisSet **intermediate\_**

*Intermediate Basis set.*

- SharedLocalizer **localizer\_**

*Molecular Orbital Localizer.*

- std::string **name\_**

*Name of this OEP;.*

- std::map< std::string, **OEPType** > **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: Canonical (CMO)*

- `std::shared_ptr< psi::Matrix > cVir_`

*Virtual orbitals.*

- `std::shared_ptr< psi::Matrix > lOcc_`

*Occupied orbitals: Localized (LMO)*

- `std::vector< std::shared_ptr< psi::Vector > > lmoc_`

*LMO Centroids.*

## 18.102.1 Detailed Description

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

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

## 18.102.2 Constructor & Destructor Documentation

### 18.102.2.1 OEPotential() [1/2]

```
OEPotential::OEPotential (
    SharedWavefunction wfn,
    Options & options )
```

#### Parameters

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

**18.102.2.2 OEPotential()** [2/2]

```
OEPotential::OEPotential (
    SharedWavefunction wfn,
    SharedBasisSet auxiliary,
    SharedBasisSet intermediate,
    Options & options )
```

**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

**18.102.3 Member Function Documentation****18.102.3.1 build()** [1/2]

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

**Parameters**

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

**18.102.3.2 build()** [2/2]

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



## 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

## 18.102.3.3 make\_oeps3d()

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

## 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:

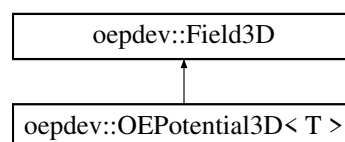
- oepdev/liboep/[oep.h](#)
- oepdev/liboep/oep\_base.cc

## 18.103 oepdev::OEPotential3D&lt; T &gt; 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

### 18.103.1 Detailed Description

template<class T>

class oepdev::OEPotential3D< T >

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

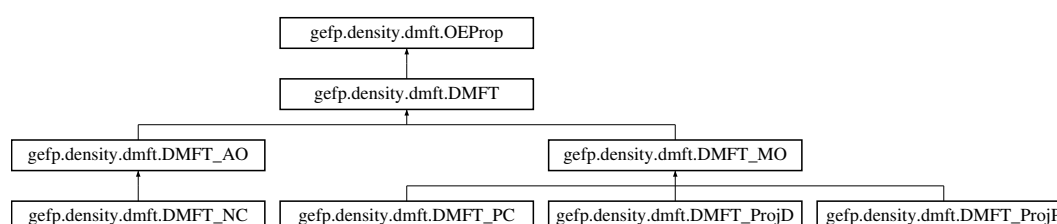
<i>T</i>	the compatible class (e.g. <code>oepdev::OEPotential</code> )
----------	---

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

- `oepdev/lib3d/space3d.h`

## 18.104 gefp.density.dmft.OEProp Class Reference

Inheritance diagram for gefp.density.dmft.OEProp:



## Static Public Member Functions

- `def dipole_moment (dmft)`
- `def quadrupole_moment (dmft)`

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

- `gefp/gefp/density/dmft.py`

## 18.105 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 [dmtip](#)

*Distributed Multipole Object.*

- SharedCISData [cis\\_data](#)

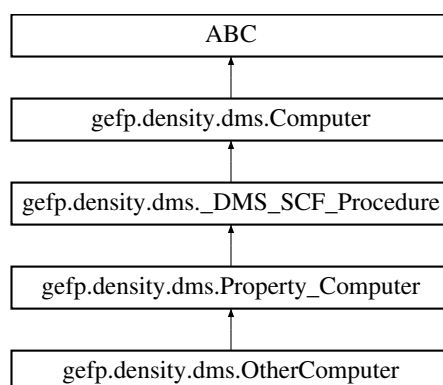
*CIS data.*

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

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

## 18.106 `gefp.density.dms.OtherComputer` Class Reference

Inheritance diagram for `gefp.density.dms.OtherComputer`:



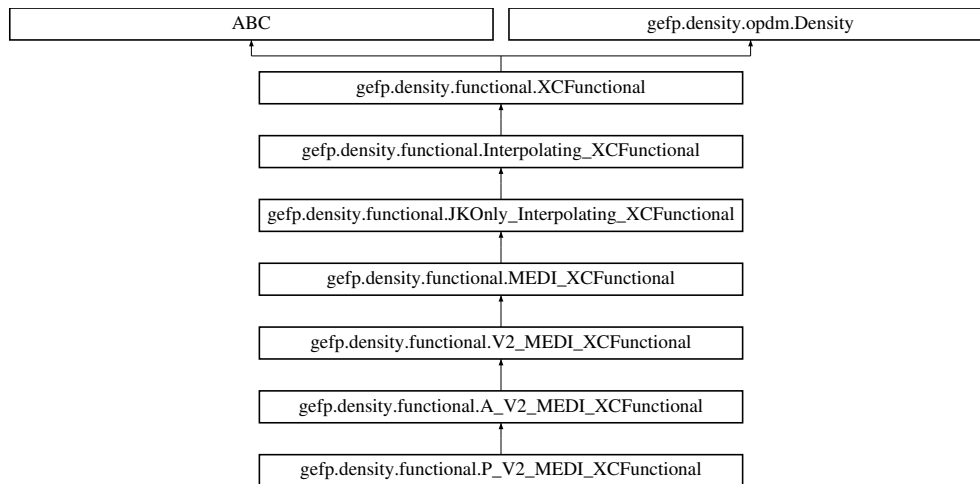
### Additional Inherited Members

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

- [gefp/gefp/density/dms.py](#)

## 18.107 `gefp.density.functional.P_V2_MEDI_XCFunctional` Class Reference

Inheritance diagram for `gefp.density.functional.P_V2_MEDI_XCFunctional`:



## Public Member Functions

- `def __init__ (self, coeff, kmax)`
- `def abbr (self)`
- `def compute_t (self, n=None, c=None)`

## Static Public Member Functions

- `def name ()`

## Additional Inherited Members

### 18.107.1 Detailed Description

The New Class of Exchange-Correlation Functionals:

Interpolation Functionals with Monotonous Exponential Decay: 2D Pade Approximant for Un

Functional coefficients:

- `'pade_coefficients'` - 2D Pade Coefficients that fit the `'t'` parameter.

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

- `gefp/gefp/density/functional.py`

## 18.108 gefp.core.driver.PadeApproximant\_2D Class Reference

## Public Member Functions

- `def __init__ (self)`
- `def add_a (self, nx, ny, a)`

- def **add\_b** (self, nx, ny, b)
- def **value** (self, x, y)

## Public Attributes

- **a**
- **b**

### 18.108.1 Detailed Description

\

Pade Approximant of Function of 2 Variables:

$$f(x, y) = \frac{\sum_{n=0}^{\infty} \sum_{m=0}^{\infty} a_{nm} x^n y^m}{1.0 + \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} b_{nm} x^n y^m} \quad \text{for } (n, m) \neq (0, 0)$$

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

- gefp/gefp/core/driver.py

## 18.109 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.*

### 18.109.1 Detailed Description

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

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

## 18.110 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:

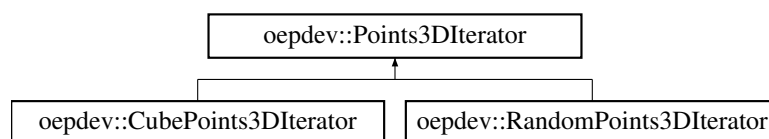
- oepdev/lib3d/space3d.h

## 18.111 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 void [rewind](#) ()  
*Rewind to the beginning.*

- 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\\_](#)

### 18.111.1 Detailed Description

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.



## 18.111.2 Constructor & Destructor Documentation

### 18.111.2.1 Points3DIterator()

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

#### Parameters

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

## 18.111.3 Member Function Documentation

### 18.111.3.1 build() [1/3]

```
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]
```

The points are generated according to Gaussian cube file format.

#### Parameters

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

**18.111.3.2 build()** [2/3]

```
std::shared_ptr< Points3DIterator > oepdev::Points3DIterator::build (
    const int & np,
    const double & radius,
    const double & cx,
    const double & cy,
    const double & cz ) [static]
```

The points are drawn according to uniform distribution 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

**18.111.3.3 build()** [3/3]

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

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:

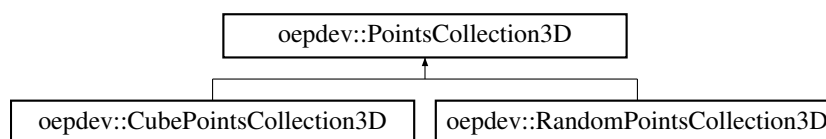
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

**18.112 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*

### 18.112.1 Detailed Description

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

**Note:** Always create instances by using static factory methods.

### 18.112.2 Constructor & Destructor Documentation

#### 18.112.2.1 PointsCollection3D()

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

#### Parameters

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

### 18.112.3 Member Function Documentation

#### 18.112.3.1 build() [1/3]

```
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]
```

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

## 18.112.3.2 build() [2/3]

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

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

## 18.112.3.3 build() [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]
```

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

## Parameters

<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

## 18.113 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

### 18.113.1 Detailed Description

Basic interface for the polarization density matrix susceptibility parameters.

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

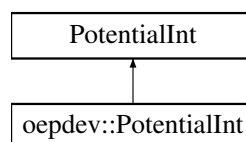
- `oepdev/libgefp/gefp.h`
- `oepdev/libgefp/gefp_polar_base.cc`

## 18.114 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::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.*

- [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.*

## 18.114.1 Constructor & Destructor Documentation

### 18.114.1.1 PotentialInt() [1/3]

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

#### 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

### 18.114.1.2 PotentialInt() [2/3]

```
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 )
```



## 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

## 18.114.1.3 PotentialInt() [3/3]

```

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 )

```

## 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

## 18.114.2 Member Function Documentation

## 18.114.2.1 set\_charge\_field()

```

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

```

## Parameters

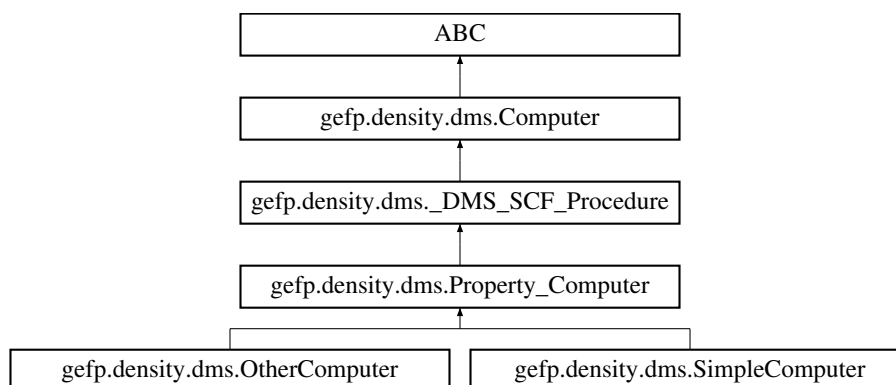
$x$	- x coordinate of $q$
$y$	- y coordinate of $q$
$z$	- z coordinate of $q$
$q$	- value of the probe charge

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

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

## 18.115 gefp.density.dms.Property\_Computer Class Reference

Inheritance diagram for gefp.density.dms.Property\_Computer:



### Public Member Functions

- def **\_\_init\_\_** (self, aggregate, fragments)
- def **atomic\_charges** (self, kappa=0.0)
- def **camm** (self)
- def **dipole\_moment** (self)
- def **ff\_dipole\_moment** (self, step=0.001, verbose=None)
- def **ff\_polarizability** (self, step=0.001, energy=False, verbose=None)
- def **ff\_hyperpolarizability** (self, step=0.001, energy=False, verbose=None)
- def **ff\_hyper2polarizability** (self, step=0.001, energy=False, verbose=None)

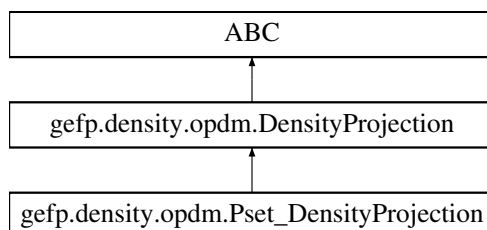
### Additional Inherited Members

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

- gefp/gefp/density/dms.py

## 18.116 gefp.density.opdm.Pset\_DensityProjection Class Reference

Inheritance diagram for gefp.density.opdm.Pset\_DensityProjection:



### Public Member Functions

- `def __init__ (self, np, S)`

### Additional Inherited Members

#### 18.116.1 Detailed Description

\  
 Gradient Projection Algorithm on P-sets.  
 Ref.: Pernal, Cancès, J. Chem. Phys. 2005

Notes:

- Appropriate for any DMFT functional.

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

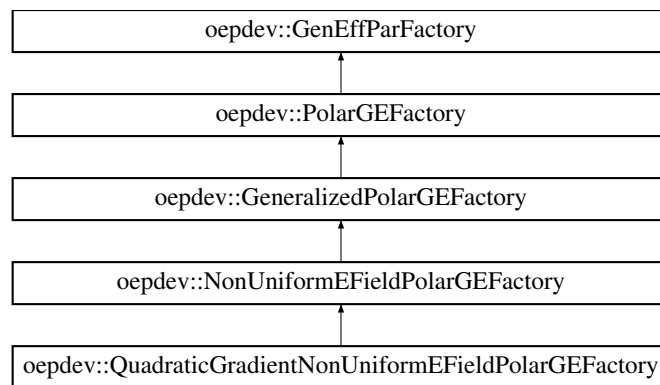
- `gefp/gefp/density/opdm.py`

## 18.117 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

### 18.117.1 Detailed Description

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  $\mathbf{r}_i$ .

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

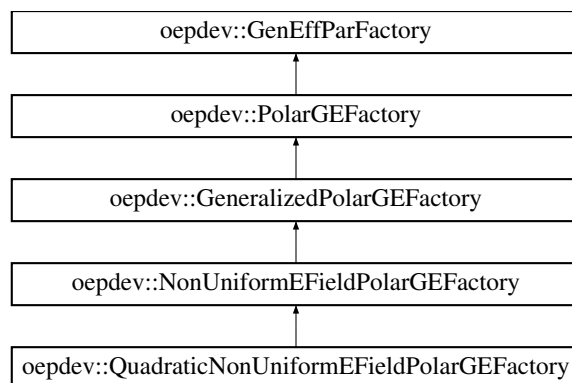
- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp\_polar\_nonuniform\_field\_2\_grad\_1.cc

## 18.118 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

#### 18.118.1 Detailed Description

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:

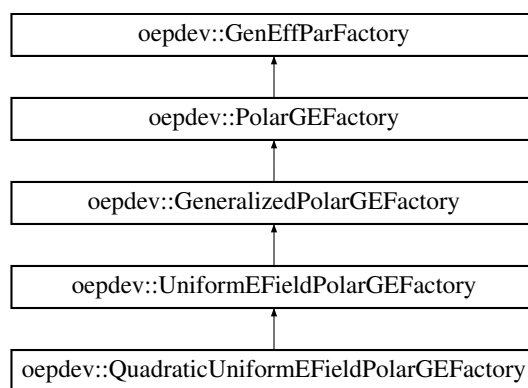
- [oepdev/libgefp/gefp.h](#)
- [oepdev/libgefp/gefp\\_polar\\_nonuniform\\_field\\_2.cc](#)

## 18.119 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

#### 18.119.1 Detailed Description

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:

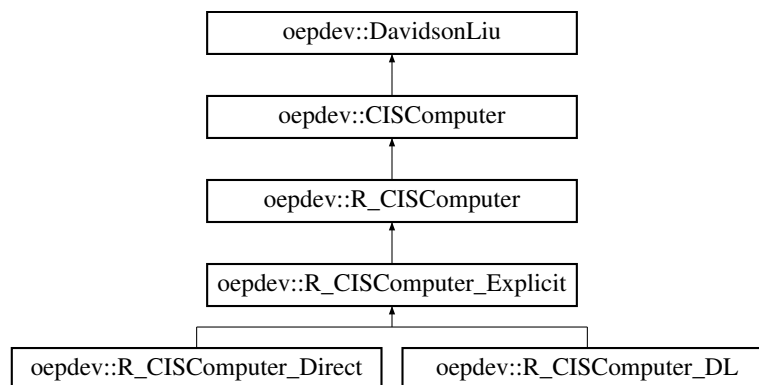
- $\mathbf{B}_{\alpha\beta}^{(10)}$  is the density matrix dipole polarizability
- $\mathbf{B}_{\alpha\beta}^{(20)}$  is the density matrix dipole-dipole hyperpolarizability

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

- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp\_polar\_uniform\_field\_2.cc

## 18.120 oepdev::R\_CISComputer Class Reference

Inheritance diagram for oepdev::R\_CISComputer:



### Public Member Functions

- **R\_CISComputer** (std::shared\_ptr< psi::Wavefunction > wfn, psi::Options &opt)

### Protected Member Functions

- virtual void **print\_excited\_state\_character\_** (int l)

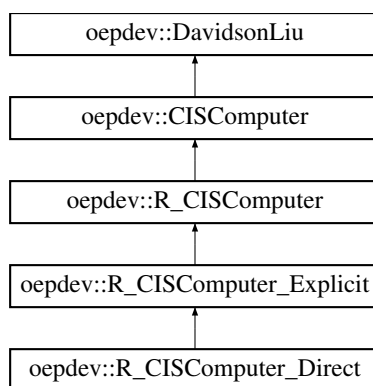
### Additional Inherited Members

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

- oepdev/libutil/[cis.h](#)
- oepdev/libutil/cis\_rhf.cc

## 18.121 oepdev::R\_CISComputer\_Direct Class Reference

Inheritance diagram for oepdev::R\_CISComputer\_Direct:



## Public Member Functions

- **R\_CISComputer\_Direct** (`std::shared_ptr< psi::Wavefunction > wfn`, `psi::Options &opt`)

## Protected Member Functions

- virtual void **build\_hamiltonian\_** (void)
- virtual void **transform\_integrals\_** (void)

## Additional Inherited Members

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

- `oepdev/libutil/cis.h`
- `oepdev/libutil/cis_rhf_direct.cc`

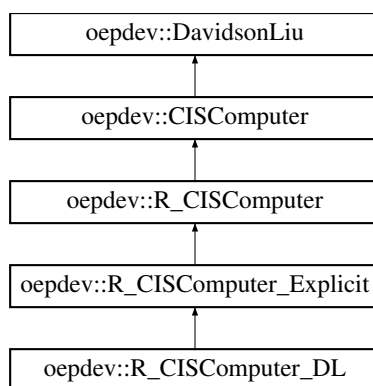
## 18.122 oepdev::R\_CISComputer\_DL Class Reference

CIS Computer with RHF reference: Davidson-Liu Solver.

```
#include <cis.h>
```

Inheritance diagram for `oepdev::R_CISComputer_DL`:





## Public Member Functions

- **R\_CISComputer\_DL** (std::shared\_ptr< psi::Wavefunction > wfn, psi::Options &opt)

## Protected Member Functions

- virtual void **set\_nstates\_** (void)
- virtual void **transform\_integrals\_** (void)
- virtual void **allocate\_hamiltonian\_** (void)
- virtual void **build\_hamiltonian\_** (void)
- virtual void **diagonalize\_hamiltonian\_** (void)
- virtual void **davidson\_liu\_compute\_diagonal\_hamiltonian** (void)
- virtual void **davidson\_liu\_compute\_sigma** (void)

## Additional Inherited Members

### 18.122.1 Detailed Description

Associated options:

- **CIS\_TYPE** - must be set to **DAVIDSON\_LIU** (Default).
- **CIS\_SCHWARTZ\_CUTOFF** - Cutoff for Schwartz ERI screening. Default: 0.0.

## Implementation

### Diagonal Hamiltonian elements

They are computed by using direct method with Schwartz screening of AO ERI's. The implementation formula is

$$H_{ii}^{aa} = \varepsilon_a - \varepsilon_i + \sum_{\alpha\beta\gamma\delta} (\alpha\beta|\gamma\delta) C_{\alpha i} C_{\delta a} (C_{\beta a} C_{\gamma i} - C_{\beta i} C_{\gamma a})$$

The block associated with beta spin is equal to alpha block.

## Sigma vectors

The sigma vectors are computed from

$$\begin{aligned}\sigma_i^{a,k} &= (\epsilon_a - \epsilon_i) b_i^{a,k} + J_i^a(\mathbf{T}^{(k)}) + J_i^a(\overline{\mathbf{T}}^{(k)}) - K_i^a(\mathbf{T}^{(k)}) \\ \sigma_i^{\bar{a},k} &= (\epsilon_a - \epsilon_i) b_i^{\bar{a},k} + J_i^a(\mathbf{T}^{(k)}) + J_i^a(\overline{\mathbf{T}}^{(k)}) - K_i^a(\overline{\mathbf{T}}^{(k)})\end{aligned}$$

where  $k$  labels the vectors and where the generalized one-particle density matrices are defined by

$$\begin{aligned}T_{\gamma\delta}^{(k)} &= \sum_{jb} C_{\delta b} b_j^{b,k} C_{\gamma j} \\ \overline{T}_{\gamma\delta}^{(k)} &= \sum_{\bar{j}\bar{b}} C_{\delta\bar{b}} \bar{b}_{\bar{j}}^{\bar{b},k} C_{\gamma\bar{j}}\end{aligned}$$

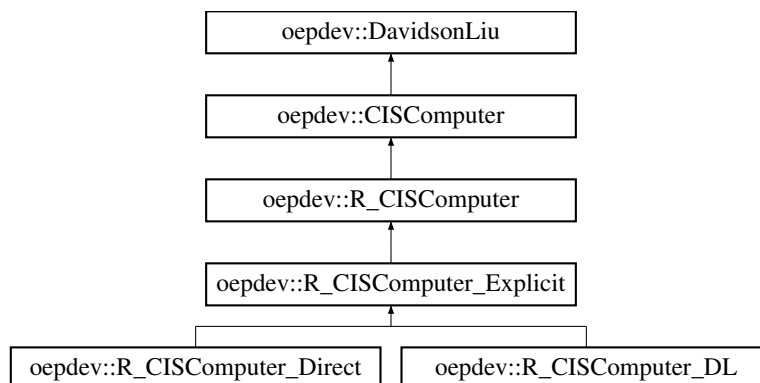
The **J** and **K** matrices in AO basis are computed by using the `psi::JK` object, and subsequently transformed to CMO's.

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

- `oepdev/libutil/cis.h`
- `oepdev/libutil/cis_rhf_dl.cc`

## 18.123 oepdev::R\_CISComputer\_Explicit Class Reference

Inheritance diagram for `oepdev::R_CISComputer_Explicit`:



## Public Member Functions

- **R\_CISComputer\_Explicit** (`std::shared_ptr< psi::Wavefunction > wfn`, `psi::Options &opt`)

## Protected Member Functions

- virtual void **set\_beta\_** (void)
- virtual void **build\_hamiltonian\_** (void)

## Additional Inherited Members

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

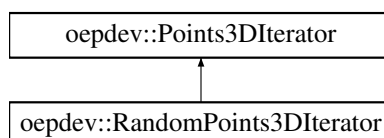
- oepdev/libutil/[cis.h](#)
- oepdev/libutil/cis\_rhf\_explicit.cc

## 18.124 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

### 18.124.1 Detailed Description

**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:

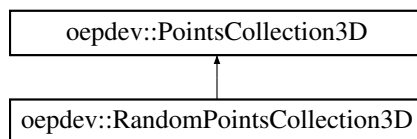
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

## 18.125 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

### 18.125.1 Detailed Description

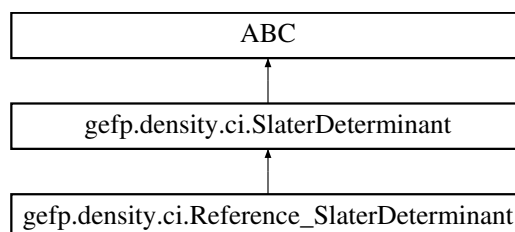
**Note:** Do not use constructors of this class explicitly. Instead, use static factory methods of the superclass to create instances.

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

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

## 18.126 gefp.density.ci.Reference\_SlaterDeterminant Class Reference

Inheritance diagram for gefp.density.ci.Reference\_SlaterDeterminant:



## Public Member Functions

- def **\_\_init\_\_** (self, nao, nbo, nmo)

## Public Attributes

- **is\_reference**

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

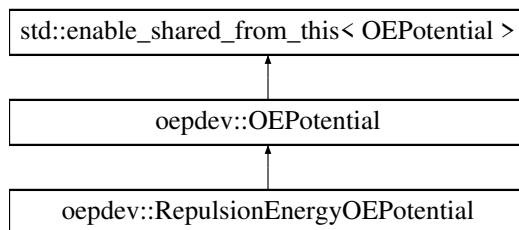
- gefp/gefp/density/ci.py

## 18.127 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  
*Header information.*

## Additional Inherited Members

### 18.127.1 Detailed Description

Contains the following OEP types:

- Murrell-etal.S1
- Otto-Ladik.S2

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

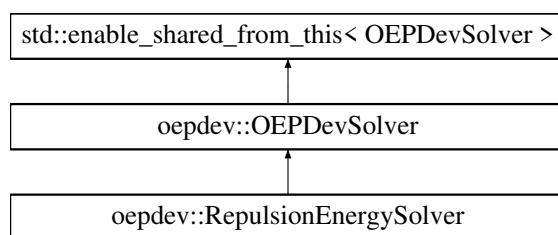
- oepdev/liboep/[oep.h](#)
- oepdev/liboep/oep\_energy\_pauli.cc

## 18.128 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

### 18.128.1 Detailed Description

The implemented methods are shown below

Table 18.107: Methods available in the Solver

Keyword	Method Description
<b>Benchmark Methods</b>	
HAYES_STONE	<i>Default.</i> Pauli Repulsion energy at HF level from Hayes and Stone (1984).
DDS	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_GDF_ESP	<i>Default.</i> OEP-Murrell et al's: S1 term via DF-OEP, S2 term via ESP-OEP.
MURRELL_ETAL_GDF_CAMM	OEP-Murrell et al's: S1 term via DF-OEP, S2 term via CAMM-OEP.

Keyword	Method Description
MURRELL_ETAL_ESP	OEP-Murrell et al's: S1 and S2 via ESP-OEP (not implemented)

*Note:*

- 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

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

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

$$\begin{aligned} E^{\text{Rep},1} &= E^{\text{Rep},\text{Kin}} + E^{\text{Rep},\text{Nuc}} \\ E^{\text{Rep},2} &= E^{\text{Rep},\text{el}-\Delta} + E^{\text{Rep},\Delta-\Delta} \end{aligned}$$

The kinetic and nuclear contributions are

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

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

$$\begin{aligned} E^{\text{Rep},\text{el}-\Delta} &= 4 \sum_{\alpha\beta\gamma\delta \in A,B} \Delta D_{\alpha\beta}^{\text{Pauli}} D_{\gamma\delta} (\alpha\beta|\gamma\delta) \\ E^{\text{Rep},\Delta-\Delta} &= 2 \sum_{\alpha\beta\gamma\delta \in A,B} \Delta D_{\alpha\beta}^{\text{Pauli}} \Delta D_{\gamma\delta}^{\text{Pauli}} (\alpha\beta|\gamma\delta) \end{aligned}$$

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

$$G_{\xi a}^A = \sum_{\xi' \in A} [\mathbf{S}^{-1}]_{\xi\xi'} \sum_{\alpha \in A} \left\{ C_{\alpha a} V_{\alpha\xi'}^A + \sum_{\mu \nu \in A} [2C_{\alpha a} D_{\mu\nu} - C_{\nu a} D_{\alpha\mu}] (\alpha\xi'|\mu\nu) \right\}$$

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

$$v_a^A(\mathbf{r}) = \sum_{x \in A} \frac{-Z_x}{|\mathbf{r} - \mathbf{r}_x|} + 2 \sum_{c \in A} \int \frac{\phi_c(\mathbf{r}') \phi_c(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' - \frac{1}{2} \int \frac{\phi_a(\mathbf{r}') \phi_a(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}'$$

## 18.128.2 Member Function Documentation

### 18.128.2.1 compute\_benchmark()

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

Each solver object has one `DEFAULT` benchmark method

#### Parameters

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

Implements [oepdev::OEPDevSolver](#).

### 18.128.2.2 compute\_oep\_based()

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

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:

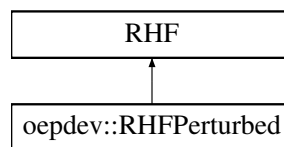
- oepdev/libsolver/[solver.h](#)
- oepdev/libsolver/solver\_energy\_pauli.cc

## 18.129 oepdev::RHFPerturbed Class Reference

RHF theory under electrostatic perturbation.

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

Inheritance diagram for oepdev::RHFPerturbed:



### Public Member Functions

- [RHFPerturbed](#) (std::shared\_ptr< psi::Wavefunction > ref\_wfn, std::shared\_ptr< psi::SuperFunctional > functional)  
*Build from wavefunction and superfunctional.*
- [RHFPerturbed](#) (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 [~RHFPerturbed](#) ()  
*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.*

### 18.129.1 Detailed Description

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.,

$$E_{\text{Nuc}} \rightarrow E_{\text{Nuc-Nuc}} + \sum_{In} \frac{q_n Z_I}{r_{In}} - \mu_{\text{Nuc}} \cdot \mathbf{F}$$

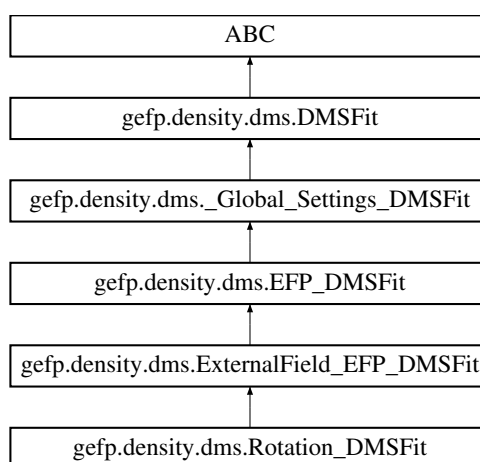
where  $\mu_{\text{Nuc}}$  is the nuclear dipole moment and  $Z_I$  is the atomic number of the  $I$ th nucleus. It is added in the nuclear repulsion energy  $E_{\text{Nuc}-\text{Nuc}}$  (note that the resulting energy can be negative as well depending on the electric field direction and configuration of point charges).

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

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

## 18.130 gefp.density.dms.Rotation\_DMSFit Class Reference

Inheritance diagram for gefp.density.dms.Rotation\_DMSFit:



### Public Member Functions

- `def __init__ (self, mol, method, nsamples, dms_types, order_type, use_iterative_model, use_external_field_model, start=1.0, srangle=2.0)`

### Additional Inherited Members

#### 18.130.1 Detailed Description

Rotation method to fit DMS tensors.

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

- `gefp/gefp/density/dms.py`

## 18.131 gefp.density.rvs.RVS Class Reference

## Public Member Functions

- `def __init__(self, wfn)`
- `def run_dimer(self, conver=1.0e-7, maxiter=100, ndamp=10)`
- `def __repr__(self)`

## Public Attributes

- `vars`

### 18.131.1 Detailed Description

---

Reduced Variational Scheme (RVS)  
 Ref.: Stevens & Fink, Chem. Phys. Lett., Vol. 139, pp. 15-22 (1987)

Implementation for closed-shell systems.

---

Notes:

o This code contains general implementation of the RVS-SCF method. Currently, only dimers are automatically analyzed. To use for multimers, define a subclass and redefine the 'run' method to include n-mers for n>2.

o Projection of frozen orbitals is achieved by transforming Fock matrix to orthogonal MO basis of entire n-fragment aggregate, and setting all the off-diagonal matrix elements that are associated to the frozen orbitals to zero [Kairys & Jensen, J. Phys. Chem. A, Vol. 104, No. 28, 2000].

o Orthogonal MO basis is constructed from the original mutually non-orthogonal MOs of isolated fragments by GrammSchmidt orthogonalization with respect to the frozen MOs. If no frozen MOs are requested, symmetric Lowdin orthogonalization is performed instead.

---

Usage for dimers:

```
e, wfn = psi4.energy('scf', return_wfn=True)
rvs = RVS(wfn)
rvs.run_dimer(conver=1.0e-7, maxiter=100, ndamp=10)
print(rvs)
# access variables:
print(rvs.vars.keys())
```

---

Example for redefining for multimers:

```
class Trimer_RVS(RVS):
    def __init__(self, wfn):
        super(RVS, self).__init__(wfn)
```

---



```

def run_trimer(self, conver, maxiter, ndamp):
    "Here there is your implementation for trimer"

    # example for A-B-C (0-1-2) trimer:

    # energy of Hartree product
    #
    # frozen_occ active_vir exclude_occ
    E_Ao_Bo_Co = self._scf([ ], [ ], [ ], conver, maxiter, damp=0.0, ndamp=ndamp)

    # energy of Bocc frozen, virtual space composed of Cvir and Bvir, and A molecule e
    E_fBo_Bv_Cv = self._scf([1], [1,2], [0], conver, maxiter, damp=0.0, ndamp=ndamp)

    # energy of Bocc and Cocc frozen and all virtual space included, Aocc active
    E_fBo_fCo_Av_Bv_Cv= self._scf([1,2], [0,1,2], [ ], conver, maxiter, damp=0.0, ndamp=ndamp)

def __repr__(self):
    "Print the contents"
    log = ''
    # ...
    return str(log)

```

---

B. B\_lasiak

Gundelfingen, 14.02.2020

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

- `gefp/gefp/density/rvs.py`

## 18.132 gefp.density.dfi.SCF Class Reference

### Public Member Functions

- `def __init__(self, mol, bfs=None)`
- `def run(self, maxit=30, conv=1.0e-7, guess=None, damp=0.01, ndamp=10, verbose=True, v_ext=None)`

### Public Attributes

- `e_nuc`  
*Accessors nuclear repulsion energy.*
- `E`
- `D`
- `Co`
- `C`
- `F`
- `eps`
- `H`
- `S`
- `X`

### 18.132.1 Detailed Description

---

Self-Consistent Field (SCF) Procedure for Hartree-Fock Model

---

Demo for RHF-SCF method (closed shells). Implements SCF algorithm with primitive damping of the AO Fock matrix.

Usage:

```
scf = SCF(molecule)
scf.run(maxit=30, conv=1.0e-7, guess=None, damp=0.01, ndamp=10, verbose=True)
```

The above example runs SCF on 'molecule' psi4.core.Molecule object starting from core Hamiltonian as guess (guess=None) and convergence 1.0E-7 A.U. in total energy with 30 maximum iterations (10 of which are performed by damping of the Fock matrix with damping coefficient of 0.1). The SCF iterations are printed to standard output (verbose=True).

---

Last Revision: Gund

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

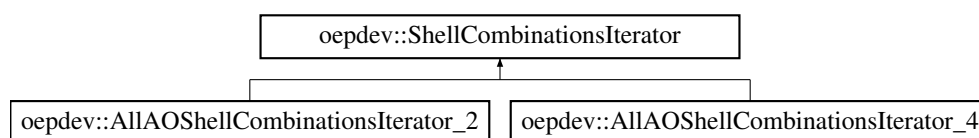
- gefp/gefp/density/dfi.py

## 18.133 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.*

### 18.133.1 Detailed Description

Date

2018/03/01 17:22:00

### 18.133.2 Constructor & Destructor Documentation

#### 18.133.2.1 ShellCombinationsIterator()

```
ShellCombinationsIterator::ShellCombinationsIterator (
    int nshell )
```

Parameters

<i>nshell</i>	- number of shells this iterator is for
---------------	---

### 18.133.3 Member Function Documentation

#### 18.133.3.1 ao\_iterator()

```
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

#### 18.133.3.2 build() [1/2]

```
std::shared_ptr< ShellCombinationsIterator > ShellCombinationsIterator::build
(
    const IntegralFactory & ints,
    std::string mode = "ALL",
    int nshell = 4 ) [static]
```

#### 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](http://example_integrals_iter.cc).

#### 18.133.3.3 build() [2/2]

```
std::shared_ptr< ShellCombinationsIterator > ShellCombinationsIterator::build
(
    const psi::IntegralFactory & ints,
    std::string mode = "ALL",
    int nshell = 4 ) [static]
```

**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

**18.133.3.4 compute\_shell()** [1/2]

```
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](#).

**18.133.3.5 compute\_shell()** [2/2]

```
void ShellCombinationsIterator::compute_shell (
    std::shared_ptr< psi::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
------------	----------------------------

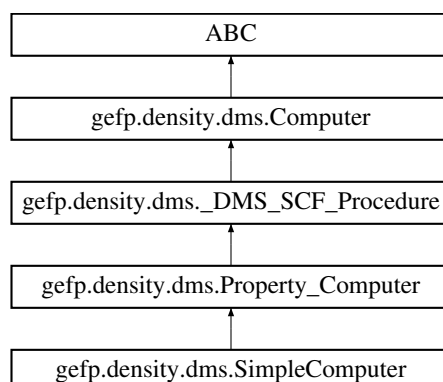
Implemented in [oepdev::AllAOShellCombinationsIterator\\_4](#).

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

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

## 18.134 gefp.density.dms.SimpleComputer Class Reference

Inheritance diagram for gefp.density.dms.SimpleComputer:



### Public Member Functions

- `def __init__ (self, args, kwargs)`

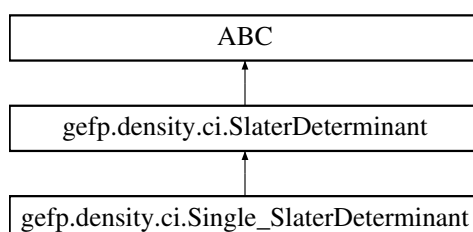
### Additional Inherited Members

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

- `gefp/gefp/density/dms.py`

## 18.135 gefp.density.ci.Single\_SlaterDeterminant Class Reference

Inheritance diagram for gefp.density.ci.Single\_SlaterDeterminant:



### Public Member Functions

- `def __init__ (self, nao, nbo, nmo, rule)`

## Public Attributes

- **is\_single**
- **change\_alpha**

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

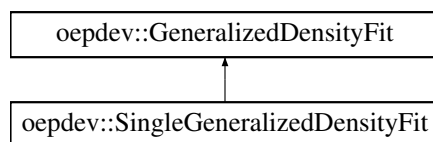
- `gefp/gefp/density/ci.py`

## 18.136 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

### 18.136.1 Detailed Description

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.

### 18.136.2 Determination of the OEP matrix

Coefficients  $\mathbf{G}$  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,  $|$  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  $\hat{v}$ .

### 18.136.3 Member Function Documentation

#### 18.136.3.1 compute()

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

#### Returns

The OEP coefficients  $G_{\xi i}$

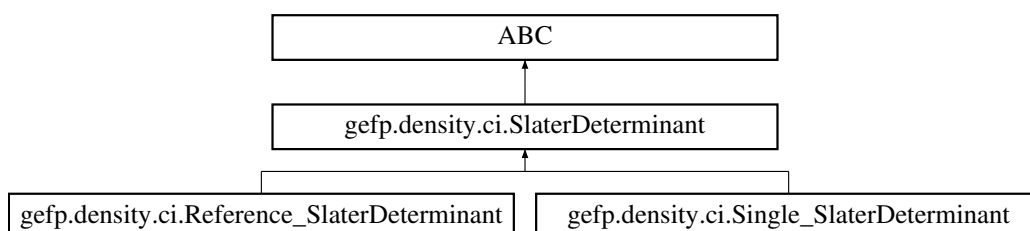
Implements [oepdev::GeneralizedDensityFit](#).

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

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

## 18.137 gefp.density.ci.SlaterDeterminant Class Reference

Inheritance diagram for gefp.density.ci.SlaterDeterminant:



## Public Member Functions

- `def __init__ (self, nao, nbo, nmo, rule)`

## Public Attributes

- `is_reference`
- `is_single`
- `is_double`
- `is_triple`
- `rule`
- `nao`
- `nbo`
- `nav`
- `nbv`
- `nmo`

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

- `gefp/gefp/density/ci.py`

## 18.138 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.*

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

- `oepdev/libgefp/gefp.h`

## 18.139 gefp.math.matrix.Superimposer Class Reference

### Public Member Functions

- def `__init__` (self)  
*SVDSuperimposer from BIOPYTHON PACKAGE Copyright (C) 2002, Thomas Hamelryck (thamelry@vub.ac.be) This code is part of the Biopython distribution and governed by its license.*
- def `set` (self, reference\_coords, coords)
- def `run` (self)
- def `get_transformed` (self)
- def `get_rotran` (self)
- def `get_init_rms` (self)
- def `get_rms` (self)

### Public Attributes

- `reference_coords`
- `coords`
- `transformed_coords`
- `rot`
- `tran`
- `rms`
- `init_rms`
- `n`

### 18.139.1 Detailed Description

\

SVDSuperimposer finds the best rotation and translation to put two point sets on top of each other (minimizing the RMSD). This is eg. useful to superimpose crystal structures.

SVD stands for Singular Value Decomposition, which is used to calculate the superposition.

Reference:

Matrix computations, 2nd ed. Golub, G. & Van Loan, CF., The Johns Hopkins University Press, Baltimore, 1989

### 18.139.2 Constructor & Destructor Documentation

**18.139.2.1 `__init__()`**

```
def gefp.math.matrix.Superimposer.__init__ (
    self )
```

Please see the LICENSE file that should have been included as part of this package.

**18.139.3 Member Function Documentation****18.139.3.1 `set()`**

```
def gefp.math.matrix.Superimposer.set (
    self,
    reference_coords,
    coords )
```

Set the coordinates to be superimposed.  
coords will be put on top of reference\_coords.

- o reference\_coords: an NxDIM array
- o coords: an NxDIM array

DIM is the dimension of the points, N is the number of points to be superimposed.

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

- gefp/gefp/math/matrix.py

**18.140 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\\_basis\\_rotation](#) (void)  
*Test the AO basis set rotation from [oepdev::ao\\_rotation\\_matrix](#).*
- double [test\\_cis\\_rhf](#) (void)  
*Test the CIS(RHF) method.*
- double [test\\_cis\\_uhf](#) (void)  
*Test the CIS(UHF) method.*
- double [test\\_cis\\_rhf\\_dl](#) (void)  
*Test the CIS(RHF) method with Davidson-Liu algorithm.*
- double [test\\_cis\\_uhf\\_dl](#) (void)  
*Test the CIS(UHF) method with Davidson-Liu algorithm.*
- 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\\_dmtp\\_pot\\_field](#) (void)  
*Test the [oepdev::MultipoleConvergence](#) class: potential and field calculations.*
- double [test\\_dmtp\\_energy](#) (void)  
*Test the [oepdev::DMTP](#) class for energy calculations.*
- double [test\\_efp2\\_energy](#) (void)  
*Test the [oepdev::EFP2\\_GenEffPar](#) and [oepdev::EFP2\\_Computer](#) classes.*
- double [test\\_kabsch\\_superimposition](#) (void)

- *Test the [oepdev::KabschSuperimposer](#).*
- double [test\\_dmtip\\_superimposition](#) (void)
  - *Test the [oepdev::DMTP](#) class for superimposition.*
- double [test\\_esp\\_solver](#) (void)
  - *Test the [oepdev::ESPSolver](#).*
- double [test\\_points\\_collection3d](#) (void)
  - *Test the cube file generation ([oepdev::Field3D](#) electrostatic potential and [oepdev::Points3DIterator](#) for cube collection)*
- double [test\\_ct\\_energy\\_benchmark\\_ol](#) (void)
  - *Test the Charge-transfer Energy Solver (benchmark method Otto-Ladik)*
- double [test\\_rep\\_energy\\_benchmark\\_hs](#) (void)
  - *Test the Repulsion Energy Solver: (benchmark method Hayes-Stone)*
- double [test\\_rep\\_energy\\_benchmark\\_dds](#) (void)
  - *Test the Repulsion Energy Solver: (benchmark method Density-Based - DDS/HF)*
- double [test\\_rep\\_energy\\_benchmark\\_murrell\\_etal](#) (void)
  - *Test the Repulsion Energy Solver: (benchmark method Murrell-etal)*
- double [test\\_rep\\_energy\\_oep\\_based\\_murrell\\_etal](#) (void)
  - *Test the Repulsion Energy Solver: (OEP-based method Murrell-etal)*
- double [test\\_rep\\_energy\\_benchmark\\_ol](#) (void)
  - *Test the Repulsion Energy Solver: (benchmark method Otto-Ladik)*
- double [test\\_rep\\_energy\\_benchmark\\_efp2](#) (void)
  - *Test the Repulsion Energy Solver: (benchmark method EFP2)*
- double [test\\_custom](#) (void)
  - *Test the custom code (to be deprecated)*

## Protected Attributes

- `std::shared_ptr< psi::Wavefunction > wfn_`  
*Wavefunction object.*
- `psi::Options & options_`  
*Psi4 Options.*

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

- `oepdev/libtest/test.h`
- `oepdev/libtest/basic.cc`
- `oepdev/libtest/basis_rotation.cc`
- `oepdev/libtest/camm.cc`
- `oepdev/libtest/cis_rhf_dl.cc`
- `oepdev/libtest/cis_rhf_explicit.cc`
- `oepdev/libtest/cis_uhf_dl.cc`

- oepdev/libtest/cis\_uhf\_explicit.cc
- oepdev/libtest/cphf.cc
- oepdev/libtest/ct\_energy\_benchmark\_ol.cc
- oepdev/libtest/dmatpol.cc
- oepdev/libtest/dmatpolX.cc
- oepdev/libtest/dmtp\_energy.cc
- oepdev/libtest/dmtp\_pot\_field.cc
- oepdev/libtest/dmtp\_superimposition.cc
- oepdev/libtest/efp2\_energy.cc
- oepdev/libtest/eri\_1\_1.cc
- oepdev/libtest/eri\_2\_2.cc
- oepdev/libtest/eri\_3\_1.cc
- oepdev/libtest/esp\_solver.cc
- oepdev/libtest/kabsch\_superimposition.cc
- oepdev/libtest/points\_collection3d.cc
- oepdev/libtest/rep\_energy\_benchmark\_dds.cc
- oepdev/libtest/rep\_energy\_benchmark\_efp2.cc
- oepdev/libtest/rep\_energy\_benchmark\_hs.cc
- oepdev/libtest/rep\_energy\_benchmark\_murrell\_etal.cc
- oepdev/libtest/rep\_energy\_benchmark\_ol.cc
- oepdev/libtest/rep\_energy\_oep\_based\_murrell\_etal.cc
- oepdev/libtest/scf\_perturb.cc
- oepdev/libtest/test.cc
- oepdev/libtest/test\_custom.cc
- oepdev/libtest/unitary\_optimizer.cc
- oepdev/libtest/unitary\_optimizer\_4\_2.cc

## 18.141 oepdev::TIData Class Reference

Transfer Integral EET Data.

```
#include <ti_data.h>
```

### Public Member Functions

- [TIData](#) ()  
*Constructor.*
- virtual [~TIData](#) ()  
*Destructor.*
- void [set\\_s](#) (double, double, double, double, double, double)  
*Set the overlap integrals between basis states,  $S_{ij}$ , for  $ij=12,13,14,23,24,34$ .*
- void [set\\_e](#) (double, double, double, double)

Set the diagonal exciton Hamiltonian matrix elements  $E_n$  for  $n=1,2,3,4$ .

- void [set\\_de](#) (double, double)  
Set environmental corrections  $\Delta E_1$  and  $\Delta E_2$ .
- void [set\\_trcamm\\_coupling](#) (oepdev::SharedDMTPConvergence)  
Set the convergence object for TrCAMM-based  $V^{\text{Coul},(0)}$ .
- virtual double [coupling\\_trcamm](#) (const std::string &rn)
- virtual double [coupling\\_direct](#) (void)
- virtual double [coupling\\_direct\\_coul](#) (void)
- virtual double [coupling\\_direct\\_exch](#) (void)
- virtual double [coupling\\_indirect](#) (void)
- virtual double [coupling\\_indirect\\_ti2](#) (void)
- virtual double [coupling\\_indirect\\_ti3](#) (void)
- virtual double [coupling\\_total](#) (void)
- virtual double [overlap\\_corrected](#) (const std::string &type)
- virtual double [overlap\\_corrected\\_direct](#) (void)
- virtual double [overlap\\_corrected\\_direct](#) (double v)
- virtual double [overlap\\_corrected\\_indirect](#) (double v, double s)

## Public Attributes

- [oepdev::MultipoleConvergence::ConvergenceLevel trcamm\\_convergence](#)  
Convergence object for Coulombic coupling under TrCAMM approximation.
- bool [diagonal\\_correction](#)  
Environmental correction activated?
- bool [mulliken\\_approximation](#)  
Mulliken approximation activated?
- bool [overlap\\_correction](#)  
Overlap correction activated?
- bool [trcamm\\_approximation](#)  
TrCAMM approximation activated?
- std::map< std::string, double > [v0](#)
- [oepdev::SharedDMTPConvergence v0\\_trcamm](#)  
 $V0_{\text{Coul}}$  multipole convergence.
- double [s12](#)  
Overlap matrix element between basis functions.
- double [s13](#)  
Overlap matrix element between basis functions.
- double [s14](#)



- *Overlap matrix element between basis functions.*
- double [s23](#)
- *Overlap matrix element between basis functions.*
- double [s24](#)
- *Overlap matrix element between basis functions.*
- double [s34](#)
- *Overlap matrix element between basis functions.*

- double [e1](#)
- *Diagonal Hamiltonian matrix element.*
- double [e2](#)
- *Diagonal Hamiltonian matrix element.*
- double [e3](#)
- *Diagonal Hamiltonian matrix element.*
- double [e4](#)
- *Diagonal Hamiltonian matrix element.*

- double [de1](#)
- *Environmental correction to the  $E_n$  for  $n=1,2$ .*
- double [de2](#)
- *Environmental correction to the  $E_n$  for  $n=1,2$ .*

## Protected Attributes

- double [c\\_](#)
- *Conversion factor (unused now)*

### 18.141.1 Detailed Description

Container for storing and managing TI data for EET coupling calculations, according to Fujimoto JCP 2012:

- exciton Hamiltonian matrix elements
- overlap integrals between basis states
- TrCamm EET coupling convergence object

Contains useful methods to process exciton Hamiltonian matrix elements:

- compute direct and indirect EET coupling constants
- compute overlap-corrected exciton Hamiltonian off-diagonal matrix elements
- include or exclude environmental correction in the diagonal exciton Hamiltonian
- activate TrCamm approximation of  $V^{\text{Coul},(0)}$
- activate Mulliken approximation for  $V^{\text{Exch},(0)}$  and  $V^{\text{CT},(0)}$

To activate/deactivate the various approximations and corrections listed above, set the following attributes

- `diagonal_correction`
- `mulliken_approximation`
- `overlap_correction`
- `trcamm_approximation`

to `true/false`, according to your need.

#### Example of usage.

```
{c++}
// Set up exciton Hamiltonian
TIData data = TIData();
data.set_s(S12, S13, S14, S32, S42, S34);
data.set_e(E1, E2, E3, E4);
data.set_de(E1 - E01, E2 - E02);
data.v0["COUL"] = V0_Coul;
data.v0["EXCH"] = V0_Exch;
data.v0["ET1"] = V0_ET1;
data.v0["ET2"] = V0_ET2;
data.v0["HT1"] = V0_HT1;
data.v0["HT2"] = V0_HT2;
data.v0["CT"] = V0_CT;
data.v0["EXCH_M"] = V0_Exch_M;
data.v0["CT_M"] = V0_CT_M;

// Set up approximations and corrections
data.diagonal_correction = true;
data.mulliken_approximation = false;
data.trcamm_approximation = false;
data.overlap_correction = true;

// Compute overlap-corrected indirect coupling matrix elements
double V_ET1 = data.overlap_corrected("ET1");
double V_ET2 = data.overlap_corrected("ET2");
double V_HT1 = data.overlap_corrected("HT1");
double V_HT2 = data.overlap_corrected("HT2");
double V_CT = data.overlap_corrected("CT");
double V_CT_M = data.overlap_corrected("CT_M");

// Compute final coupling contributions
double V_Coul = data.overlap_corrected("COUL");
double V_Exch = data.overlap_corrected("EXCH");
double V_Ovrl = data.overlap_corrected("OVRL");

double V_Exch_M = data.overlap_corrected("EXCH_M");

double V_TI_2 = data.coupling_indirect_ti2();
```

```
double V_TI_3 = data.coupling_indirect_ti3();

data.diagonal_correction = false;
double V0_TI_2 = data.coupling_indirect_ti2();
double V0_TI_3 = data.coupling_indirect_ti3();

data.mulliken_approximation = true;
double V0_TI_3_M = data.coupling_indirect_ti3();
data.diagonal_correction = true;
double V_TI_3_M = data.coupling_indirect_ti3();

double V_direct = V_Coul + V_Exch + V_Ovrl;
double V_indirect = V_TI_2 + V_TI_3;
```

**See also**

[oepdev::EETCouplingSolver](#)

**18.141.2 Member Function Documentation****18.141.2.1 coupling\_direct()**

```
double TIData::coupling_direct (
    void ) [virtual]
```

Compute the direct EET coupling constant.

**Returns**

$$V^{\text{Dir}} = V^{\text{Coul}} + V^{\text{Exch}} + V^{\text{Ovrl}}$$

Overlap and diagonal corrections as well as TrCAMM and Mulliken approximations for Coulomb and pure exchange parts can be set.

**18.141.2.2 coupling\_direct\_coul()**

```
double TIData::coupling_direct_coul (
    void ) [virtual]
```

Compute the direct EET coupling constant in Forster limit (Coulombic approximation)

**Returns**

$$V^{\text{Dir}} = V^{\text{Coul}}$$

Overlap correction as well as TrCAMM approximation for Coulomb coupling can be set.

**18.141.2.3 coupling\_direct\_exch()**

```
double TIData::coupling_direct_exch (
    void ) [virtual]
```

Compute the direct EET coupling constant due to pure exchange.

**Returns**

$$V^{\text{Dir}} = V^{\text{Exch}}$$

Overlap correction as well Mulliken approximation for pure exchange coupling can be set.

**18.141.2.4 coupling\_indirect()**

```
double TIData::coupling_indirect (
    void ) [virtual]
```

Compute the indirect EET coupling constant.

**Returns**

$$V^{\text{Indir}} = V^{\text{TI},(2)} + V^{\text{TI},(3)}$$

Overlap and diagonal corrections as well as Mulliken approximations for  $V^{\text{CT},(0)}$  can be set.

**18.141.2.5 coupling\_indirect\_ti2()**

```
double TIData::coupling_indirect_ti2 (
    void ) [virtual]
```

Compute the indirect EET coupling constant in second-order with respect to TI.

**Returns**

$$V^{\text{TI},(2)} = -\frac{V^{\text{ET}1}V^{\text{HT}2}}{E_3-E_1} - \frac{V^{\text{ET}2}V^{\text{HT}1}}{E_4-E_1}$$

Overlap and diagonal corrections can be set.

**18.141.2.6 coupling\_indirect\_ti3()**

```
double TIData::coupling_indirect_ti3 (
    void ) [virtual]
```

Compute the indirect EET coupling constant in third-order with respect to TI.

**Returns**

$$V^{\text{TI},(3)} = \frac{V^{\text{CT}}(V^{\text{ET}1}V^{\text{ET}2} + V^{\text{HT}1}V^{\text{HT}2})}{(E_3-E_1)(E_4-E_1)}$$

Overlap and diagonal corrections as well as Mulliken approximations for  $V^{\text{CT},(0)}$  can be set.

**18.141.2.7 coupling\_total()**

```
double TIData::coupling_total (
    void ) [virtual]
```

Compute the *total* EET coupling constant.

**Returns**

$$V^{\text{Total}} = V^{\text{Dir}} + V^{\text{Indir}}$$

Overlap and diagonal corrections, TrCAMM approximation for Coulomb coupling, and Mulliken approximations for pure exchange coupling and  $V^{\text{CT},(0)}$  can be set.

**18.141.2.8 coupling\_trcamm()**

```
double TIData::coupling_trcamm (
    const std::string & rn ) [virtual]
```

Compute Coulombic coupling approximated by TrCAMM.

**Parameters**

<i>rn</i>	- convergence of TrCAMM coupling. Can be from R1 to R5, which corresponds to the $R^{-n}$ series expansion of distributed multipoles.
-----------	---

**Returns**

$$V^{\text{Coul},(0)} \approx V^{\text{TrCAMM},(0)}(R^{-n})$$

**18.141.2.9 overlap\_corrected()**

```
double TIData::overlap_corrected (
    const std::string & type ) [virtual]
```

Compute overlap corrected matrix elements.

## Parameters

<i>type</i>	<p>- matrix element <math>V^{\text{type},(0)}</math> subject to overlap correction, where type is one of the following:</p> <ul style="list-style-type: none"> <li>• COUL - <math>V^{\text{Coul},(0)}</math>,</li> <li>• EXCH - <math>V^{\text{Exch},(0)}</math>,</li> <li>• TrCamm_R1 - <math>V^{\text{TrCamm},(0)}(R^{-1})</math>,</li> <li>• TrCamm_R2 - <math>V^{\text{TrCamm},(0)}(R^{-2})</math>,</li> <li>• TrCamm_R3 - <math>V^{\text{TrCamm},(0)}(R^{-3})</math>,</li> <li>• TrCamm_R4 - <math>V^{\text{TrCamm},(0)}(R^{-4})</math>,</li> <li>• TrCamm_R5 - <math>V^{\text{TrCamm},(0)}(R^{-5})</math>,</li> <li>• ET1 - <math>V^{\text{ET1},(0)}</math>,</li> <li>• ET2 - <math>V^{\text{ET2},(0)}</math>,</li> <li>• HT1 - <math>V^{\text{HT1},(0)}</math>,</li> <li>• HT2 - <math>V^{\text{HT2},(0)}</math>,</li> <li>• CT - <math>V^{\text{CT},(0)}</math>,</li> <li>• CT_M - Mulliken-approximated <math>V^{\text{CT},(0)}</math>,</li> <li>• EXCH_M - Mulliken-approximated <math>V^{\text{Exch},(0)}</math>.</li> </ul>
-------------	---

If type = OVRL, the overlap-correction to the direct EET coupling constant is returned,  $V^{\text{Ovrl}}$ .

## Returns

overlap-corrected exciton Hamiltonian matrix element contribution of selected type

Diagonal correction can be set.

## 18.141.2.10 overlap\_corrected\_direct() [1/2]

```
double TIData::overlap_corrected_direct (
    void ) [virtual]
```

Compute overlap-corrected direct EET coupling constant.

## Returns

$$V^{\text{Dir}} = V^{\text{Coul}} + V^{\text{Exch}} + V^{\text{Ovrl}}$$

Diagonal correction, TrCamm approximation and Mulliken approximation can be set.

**18.141.2.11 overlap\_corrected\_direct()** [2/2]

```
double TIData::overlap_corrected_direct (
    double v ) [virtual]
```

Compute overlap-corrected direct EET coupling constant from value  $v$ .

**Returns**

$$\frac{v}{1-S_{12}^2}$$

**18.141.2.12 overlap\_corrected\_indirect()**

```
double TIData::overlap_corrected_indirect (
    double v,
    double s ) [virtual]
```

Compute overlap-corrected coupling constant from value  $v$  and associated overlap integral  $s$ .

**Returns**

$$\frac{1}{1-s^2} \left( v - \frac{(E_1+E_2)s}{2} \right)$$

Diagonal correction can be set.

**18.141.3 Member Data Documentation****18.141.3.1 v0**

```
std::map<std::string, double> oepdev::TIData::v0
```

Dictionary of all zeroth-order off-diagonal matrix elements.

Use only the following keywords:

- COUL -  $V^{\text{Coul},(0)}$ ,
- EXCH -  $V^{\text{Exch},(0)}$ ,
- TrCAMM\_R1 -  $V^{\text{TrCAMM},(0)}(R^{-1})$ ,
- TrCAMM\_R2 -  $V^{\text{TrCAMM},(0)}(R^{-2})$ ,
- TrCAMM\_R3 -  $V^{\text{TrCAMM},(0)}(R^{-3})$ ,
- TrCAMM\_R4 -  $V^{\text{TrCAMM},(0)}(R^{-4})$ ,

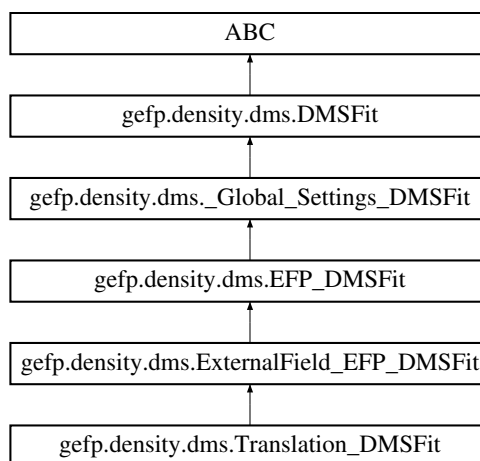
- $\text{TrCAMM\_R5} - V^{\text{TrCAMM},(0)}(R^{-5})$ ,
- $\text{ET1} - V^{\text{ET1},(0)}$ ,
- $\text{ET2} - V^{\text{ET2},(0)}$ ,
- $\text{HT1} - V^{\text{HT1},(0)}$ ,
- $\text{HT2} - V^{\text{HT2},(0)}$ ,
- $\text{CT} - V^{\text{CT},(0)}$ ,
- $\text{CT\_M}$  - Mulliken-approximated  $V^{\text{CT},(0)}$ ,
- $\text{EXCH\_M}$  - Mulliken-approximated  $V^{\text{Exch},(0)}$ ,
- $\text{OVRL} - V^{\text{Ovrl}}$ .

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

- [oepdev/libsolver/ti\\_data.h](#)
- [oepdev/libsolver/ti\\_data.cc](#)

## 18.142 gefp.density.dms.Translation\_DMSFit Class Reference

Inheritance diagram for gefp.density.dms.Translation\_DMSFit:



### Public Member Functions

- `def __init__(self, mol, method, nsamples, dms_types, order_type, use_iterative_model, use_external_field_model)`



## Additional Inherited Members

### 18.142.1 Detailed Description

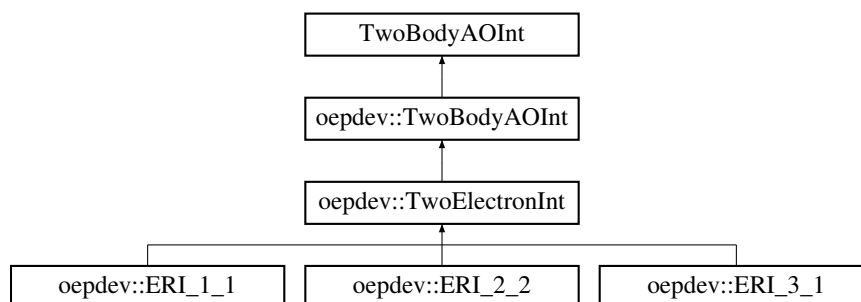
Translation method to fit DMS tensors.

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

- `gefp/gefp/density/dms.py`

## 18.143 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)

### 18.143.1 Member Function Documentation

#### 18.143.1.1 `compute()` [1/2]

```
void oepdev::TwoBodyAOInt::compute (
    std::shared_ptr< psi::Matrix > & result,
    int ibs1 = 0,
    int ibs2 = 2 ) [virtual]
```

##### 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

#### 18.143.1.2 `compute()` [2/2]

```
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:

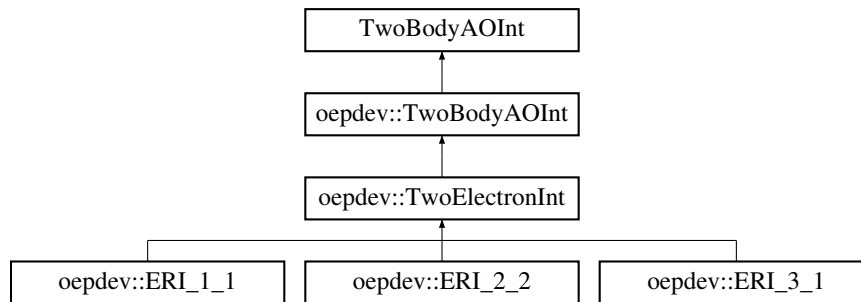
- oepdev/libpsi/[integral.h](#)
- oepdev/libpsi/integral.cc

## 18.144 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.*

### 18.144.1 Detailed Description

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](#)

### 18.144.2 Member Function Documentation

## 18.144.2.1 compute\_shell()

```
size_t oepdev::TwoElectronInt::compute_shell (
    const psi::AOShellCombinationsIterator & shellIter ) [virtual]
```

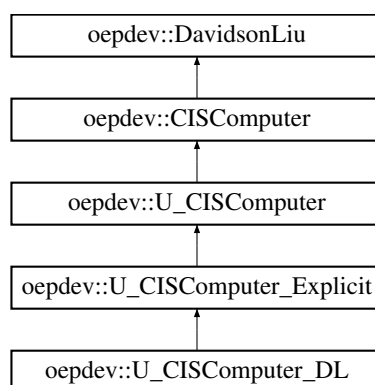
Compute ERIs between 4 shells. Result is stored in buffer. Only for use with [ERI\\_2\\_2](#) and the same basis sets, otherwise shell pairs won't be compatible.

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

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

## 18.145 oepdev::U\_CISComputer Class Reference

Inheritance diagram for oepdev::U\_CISComputer:



## Public Member Functions

- **U\_CISComputer** (std::shared\_ptr< psi::Wavefunction > wfn, psi::Options &opt)

## Protected Member Functions

- virtual void **print\_excited\_state\_character\_** (int I)

## Additional Inherited Members

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

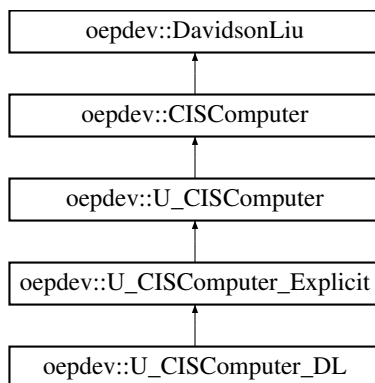
- oepdev/libutil/[cis.h](#)
- oepdev/libutil/cis\_uhf.cc

## 18.146 oepdev::U\_CISComputer\_DL Class Reference

CIS Computer with UHF reference: Davidson-Liu Solver.

```
#include <cis.h>
```

Inheritance diagram for oepdev::U\_CISComputer\_DL:



### Public Member Functions

- **U\_CISComputer\_DL** (std::shared\_ptr< psi::Wavefunction > wfn, psi::Options &opt)

### Protected Member Functions

- virtual void **set\_nstates\_** (void)
- virtual void **transform\_integrals\_** (void)
- virtual void **allocate\_hamiltonian\_** (void)
- virtual void **build\_hamiltonian\_** (void)
- virtual void **diagonalize\_hamiltonian\_** (void)
- virtual void **davidson\_liu\_compute\_diagonal\_hamiltonian** (void)
- virtual void **davidson\_liu\_compute\_sigma** (void)

### Additional Inherited Members

#### 18.146.1 Detailed Description

Associated options:

- **CIS\_TYPE** - must be set to DAVIDSON\_LIU (Default).
- **CIS\_SCHWARTZ\_CUTOFF** - Cutoff for Schwartz ERI screening. Default: 0.0.

## Implementation

### Diagonal Hamiltonian elements

They are computed by using direct method with Schwartz screening of AO ERI's. The implementation formula is

$$H_{ii}^{aa} = \varepsilon_a - \varepsilon_i + \sum_{\alpha\beta\gamma\delta} (\alpha\beta|\gamma\delta) C_{\alpha i} C_{\delta a} (C_{\beta a} C_{\gamma i} - C_{\beta i} C_{\gamma a})$$

$$H_{ii}^{\bar{a}\bar{a}} = \varepsilon_{\bar{a}} - \varepsilon_{\bar{i}} + \sum_{\alpha\beta\gamma\delta} (\alpha\beta|\gamma\delta) C_{\alpha \bar{i}} C_{\delta \bar{a}} (C_{\beta \bar{a}} C_{\gamma \bar{i}} - C_{\beta \bar{i}} C_{\gamma \bar{a}})$$

### Sigma vectors

The sigma vectors are computed from

$$\sigma_i^{a,k} = (\varepsilon_a - \varepsilon_i) b_i^{a,k} + J_i^a(\mathbf{T}^{(k)}) + J_i^a(\overline{\mathbf{T}^{(k)}}) - K_i^a(\mathbf{T}^{(k)})$$

$$\sigma_{\bar{i}}^{\bar{a},k} = (\varepsilon_{\bar{a}} - \varepsilon_{\bar{i}}) b_{\bar{i}}^{\bar{a},k} + J_{\bar{i}}^{\bar{a}}(\mathbf{T}^{(k)}) + J_{\bar{i}}^{\bar{a}}(\overline{\mathbf{T}^{(k)}}) - K_{\bar{i}}^{\bar{a}}(\overline{\mathbf{T}^{(k)}})$$

where  $k$  labels the vectors and where the generalized one-particle density matrices are defined by

$$T_{\gamma\delta}^{(k)} = \sum_{jb} C_{\delta b} b_j^{b,k} C_{\gamma j}$$

$$\bar{T}_{\gamma\delta}^{(k)} = \sum_{\bar{j}\bar{b}} C_{\delta \bar{b}} \bar{b}_{\bar{j}}^{\bar{b},k} C_{\gamma \bar{j}}$$

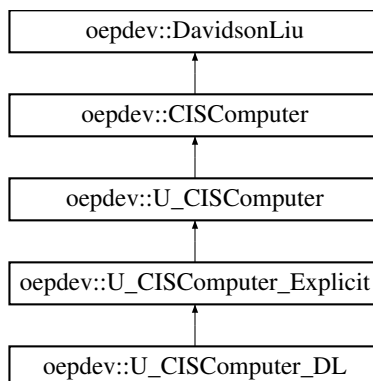
The **J** and **K** matrices in AO basis are computed by using the `psi::JK` object, and subsequently transformed to CMO's.

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

- oepdev/libutil/cis.h
- oepdev/libutil/cis\_uhf\_dl.cc

## 18.147 oepdev::U\_CISComputer\_Explicit Class Reference

Inheritance diagram for oepdev::U\_CISComputer\_Explicit:



## Public Member Functions

- **U\_CISComputer\_Explicit** (std::shared\_ptr< psi::Wavefunction > wfn, psi::Options &opt)

## Protected Member Functions

- virtual void **set\_beta\_** (void)
- virtual void **build\_hamiltonian\_** (void)

## Additional Inherited Members

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

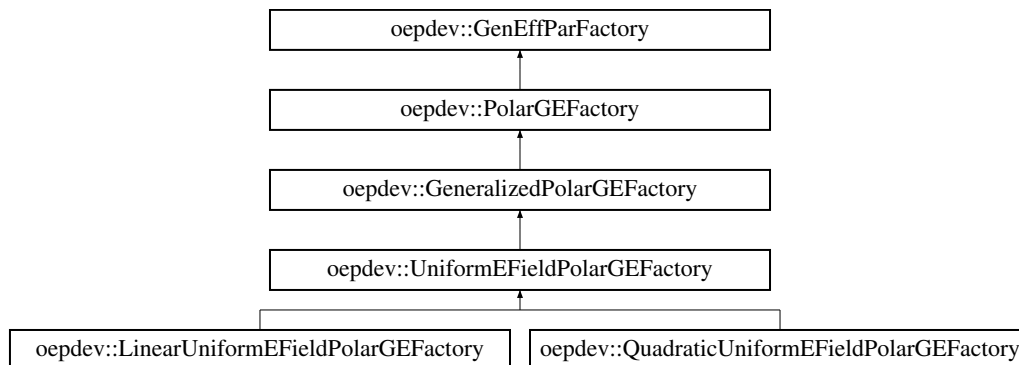
- oepdev/libutil/cis.h
- oepdev/libutil/cis\_uhf\_explicit.cc

## 18.148 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

### 18.148.1 Detailed Description

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

## 18.149 oepdev::UnitaryOptimizer Class Reference

Find the optimim 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.*

### 18.149.1 Detailed Description

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

## 18.149.2 Constructor & Destructor Documentation

### 18.149.2.1 UnitaryOptimizer() [1/3]

```
oepdev::UnitaryOptimizer::UnitaryOptimizer (
    double * R,
    double * P,
    int n,
    double conv = 1.0e-6,
    int maxiter = 100,
    bool verbose = true )
```

#### Parameters

<i>R</i>	- <b>R</b> matrix
<i>P</i>	- <b>P</b> vector
<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.

### 18.149.2.2 UnitaryOptimizer() [2/3]

```
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 )
```

#### Parameters

<i>R</i>	- <b>R</b> matrix
<i>P</i>	- <b>P</b> vector
<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.

## 18.149.2.3 UnitaryOptimizer() [3/3]

```
oepdev::UnitaryOptimizer::UnitaryOptimizer (
    int n,
    double conv,
    int maxiter,
    bool verbose ) [protected]
```

## Parameters

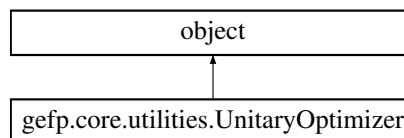
<i>n</i>	- dimensionality of the problem ( <i>N</i> )
<i>conv</i>	- convergence in the <i>Z</i> function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

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

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

## 18.150 gefp.core.utilities.UnitaryOptimizer Class Reference

Inheritance diagram for gefp.core.utilities.UnitaryOptimizer:



## Public Member Functions

- def `__init__` (self, R, P, conv=1.0e-8, maxiter=100, verbose=True)
- def [maximize](#) (self)
- def [minimize](#) (self)
- def [run](#) (self, opt='minimize')
- def [Z](#) (self)

## Public Attributes

- **X**
- **conv**
- **maxiter**
- **verbose**

### 18.150.1 Detailed Description

---

Finds the unitary matrix  $X$  that optimizes the following function:

$$Z(X) = \sum_{ijkl} X_{ij} X_{kl} R_{jl} - \sum_{ij} X_{ij} P_{ij}$$

where

- \*  $X$  is a square unitary matrix of size  $N \times N$
- \*  $R$  is a square, in general non-symmetric matrix of size  $N \times N$
- \*  $P$  is a vector of length  $N$

Usage:

```
optimizer = UnitaryOptimizer(R, P, conv=1.0e-8, maxiter=100, verbose=True)
optimizer.maximize() #or minimize()
X = optimizer.X
Z = optimizer.Z
```

---

Last Revision: 25.03.2018

### 18.150.2 Constructor & Destructor Documentation

#### 18.150.2.1 `__init__()`

```
def gefp.core.utilities.UnitaryOptimizer.__init__ (
    self,
    R,
    P,
    conv = 1.0e-8,
    maxiter = 100,
    verbose = True )
```

Initialize with  $R$  and  $P$  matrix, as well as optimization options

### 18.150.3 Member Function Documentation

#### 18.150.3.1 `maximize()`

```
def gefp.core.utilities.UnitaryOptimizer.maximize (
    self )
```

Maximize the  $Z$  function under unitary constraint for  $X$



**18.150.3.2 minimize()**

```
def gefp.core.utilities.UnitaryOptimizer.minimize (
    self )
```

Minimize the Z function under unitary constraint for X

**18.150.3.3 run()**

```
def gefp.core.utilities.UnitaryOptimizer.run (
    self,
    opt = 'minimize' )
```

Perform the optimization

**18.150.3.4 Z()**

```
def gefp.core.utilities.UnitaryOptimizer.Z (
    self )
```

Return the current value of objective function

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

- gefp/gefp/core/utilities.py

**18.151 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.*

### 18.151.1 Detailed Description

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

$$R_{ijklmn} \rightarrow \sum_{k'l'm'n'} R_{ijk'l'm'n'} X_{k'k} X_{l'l} X_{m'm} X_{n'n}$$

$$P_{ijk} \rightarrow \sum_{j'k'} P_{ij'k'} X_{j'j} X_{k'k}$$

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

## 18.151.2 Constructor & Destructor Documentation

### 18.151.2.1 UnitaryOptimizer\_4\_2() [1/2]

```
oepdev::UnitaryOptimizer_4_2::UnitaryOptimizer_4_2 (
    double * R,
    double * P,
    int n,
    double conv = 1.0e-6,
    int maxiter = 100,
    bool verbose = true )
```

#### 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.

### 18.151.2.2 UnitaryOptimizer\_4\_2() [2/2]

```
oepdev::UnitaryOptimizer_4_2::UnitaryOptimizer_4_2 (
    int n,
    double conv,
    int maxiter,
    bool verbose ) [protected]
```

#### Parameters

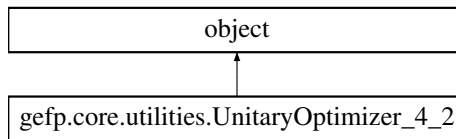
<i>n</i>	- dimensionality of the problem ( <i>N</i> )
<i>conv</i>	- convergence in the <i>Z</i> function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

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

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

## 18.152 gefp.core.utilities.UnitaryOptimizer\_4\_2 Class Reference

Inheritance diagram for gefp.core.utilities.UnitaryOptimizer\_4\_2:



### Public Member Functions

- def `__init__` (self, R, P, conv=1.0e-8, maxiter=100, verbose=True)
- def `maximize` (self)
- def `minimize` (self)
- def `run` (self, opt='minimize')
- def `Z` (self)

### Public Attributes

- **X**
- **conv**
- **maxiter**
- **verbose**

#### 18.152.1 Detailed Description

---

Finds the unitary matrix X that optimizes the following function:

$$Z(X) = \sum_{ijklmn} X_{ki} X_{lj} X_{mi} X_{nj} R_{ijklmn} + \sum_{ijk} X_{ji} X_{ki} P_{ijk}$$

where

- \* X is a square unitary matrix of size N x N
- \* R is a general real 6-th rank tensor of size N<sup>6</sup>
- \* P is a general real 3-rd rank tensor of size N<sup>3</sup>

Usage:

```

optimizer = UnitaryOptimizer(R, P, conv=1.0e-8, maxiter=100, verbose=True)
optimizer.maximize() #or minimize()
X = optimizer.X
Z = optimizer.Z
  
```

---

Last Revision: 07.04.2018

## 18.152.2 Constructor & Destructor Documentation

### 18.152.2.1 `__init__()`

```
def gefp.core.utilities.UnitaryOptimizer_4.2.__init__ (
    self,
    R,
    P,
    conv = 1.0e-8,
    maxiter = 100,
    verbose = True )
```

Initialize with R and P matrix, as well as optimization options

## 18.152.3 Member Function Documentation

### 18.152.3.1 `maximize()`

```
def gefp.core.utilities.UnitaryOptimizer_4.2.maximize (
    self )
```

Maximize the Z function under unitary constraint for X

### 18.152.3.2 `minimize()`

```
def gefp.core.utilities.UnitaryOptimizer_4.2.minimize (
    self )
```

Minimize the Z function under unitary constraint for X

### 18.152.3.3 `run()`

```
def gefp.core.utilities.UnitaryOptimizer_4.2.run (
    self,
    opt = 'minimize' )
```

Perform the optimization



## 18.152.3.4 Z()

```
def gefp.core.utilities.UnitaryOptimizer_4.2.Z (
    self )
```

Return the current value of objective function

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

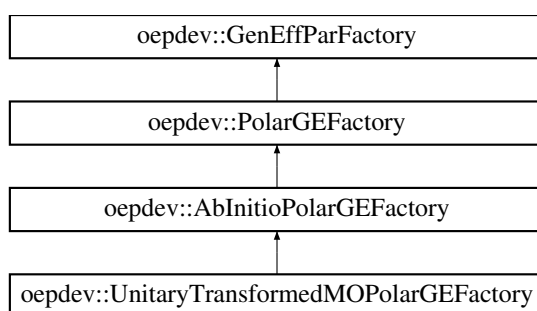
- gefp/gefp/core/utilities.py

## 18.153 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

## 18.153.1 Detailed Description

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](#)

## 18.154 `gefp.core.driver.UniversalSurface` Class Reference

### Static Public Attributes

- string `par_descr_fci_sto3g_1`
- string `par_fulld_fci_sto3g_1`
- `par_pade_fci_sto3g_1` = [PadeApproximant\\_2D\(\)](#)
- string `par_descr_fci_sto3g_2`
- string `par_fulld_fci_sto3g_2`
- `par_pade_fci_sto3g_2` = [PadeApproximant\\_2D\(\)](#)
- `par_fci_sto3g_1` = [Entry](#)(`par_pade_fci_sto3g_1`, `par_descr_fci_sto3g_1`, `par_fulld_fci_sto3g_1`)
- `par_fci_sto3g_2` = [Entry](#)(`par_pade_fci_sto3g_2`, `par_descr_fci_sto3g_2`, `par_fulld_fci_sto3g_2`)

### 18.154.1 Member Data Documentation

#### 18.154.1.1 `par_descr_fci_sto3g_1`

```
string gefp.core.driver.UniversalSurface.par_descr_fci_sto3g_1 [static]
```

#### Initial value:

```
= """\
  Universal Surface at FCI/STO-3G level.
  Systems: 2e1 (H2), 4e1 (H4), 10e1 (H2O)
  """
```

**18.154.1.2 par\_descr\_fci\_sto3g\_2**

```
string gefp.core.driver.UniversalSurface.par_descr_fci_sto3g_2 [static]
```

**Initial value:**

```
= """\
  Universal Surface at FCI/STO-3G level.
  Systems: 2e1 (H2), 4e1 (H4), 10e1 (H2O)
  """
```

**18.154.1.3 par\_fulld\_fci\_sto3g\_1**

```
string gefp.core.driver.UniversalSurface.par_fulld_fci_sto3g_1 [static]
```

**Initial value:**

```
= """\
  function used for fitting: g(x,y)
  g(x,y) = (a0 + a1*x + a2*y + a3*x*y + a4*y*y + a5*y*y*x + a6*y*y*y)/(1.0 + b1*x + b2*y + b3*x*y + b4*y*y
    + b5*y*y*x + b6*y*y*y)

  degrees of freedom      (FIT_NDF)                : 47
  rms of residuals        (FIT_STDFIT) = sqrt(WSSR/ndf) : 0.0217961
  variance of residuals   (reduced chisquare) = WSSR/ndf : 0.00047507

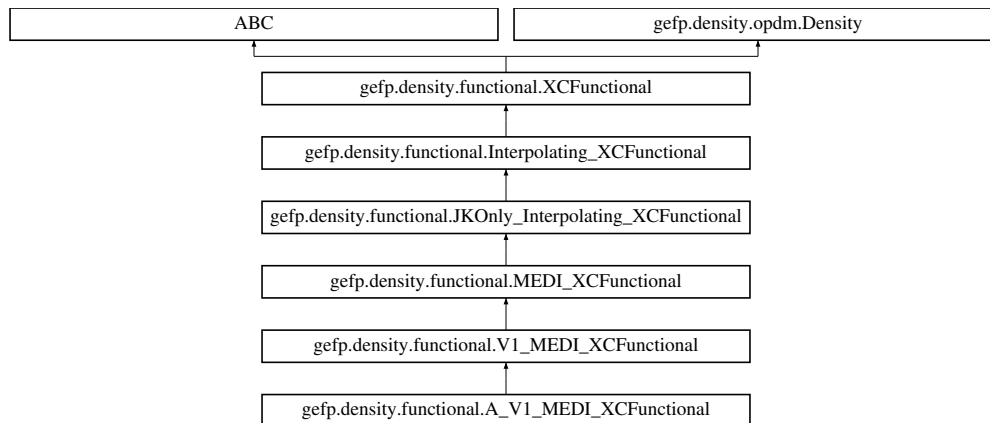
  Final set of parameters          Asymptotic Standard Error
  =====
a0      = 0.173323      +/- 0.007845      (4.526%)
a1      = -6.85743      +/- 3.626          (52.87%)
a2      = 0.732897      +/- 0.5255         (71.7%)
a3      = -8.10918      +/- 3.951          (48.73%)
a4      = -0.272454      +/- 0.5683         (208.6%)
a5      = 1.73152       +/- 3.307           (191%)
a6      = -0.0591462     +/- 0.08416        (142.3%)
b1      = -68.04         +/- 38.64          (56.8%)
b2      = -3.57443       +/- 3.66           (102.4%)
b3      = 23.0961        +/- 43.09          (186.5%)
b4      = -3.80028       +/- 3.203          (84.29%)
b5      = 30.2229        +/- 27.85          (92.14%)
b6      = -0.269245      +/- 0.328          (121.8%)
  """
```

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

- gefp/gefp/core/driver.py

**18.155 gefp.density.functional.V1\_MEDI\_XCFunctional Class Reference**

Inheritance diagram for gefp.density.functional.V1\_MEDI\_XCFunctional:



## Public Member Functions

- `def __init__ (self, coeff, kmax)`
- `def compute_ak (self, k, t)`
- `def compute_t (self, n=None, c=None)`

## Additional Inherited Members

### 18.155.1 Detailed Description

\
 Version 1 of MEDI functional. It is a proper functional of density matrix. However, it cannot be represented explicitly in terms of density matrix.

Functional coefficients:

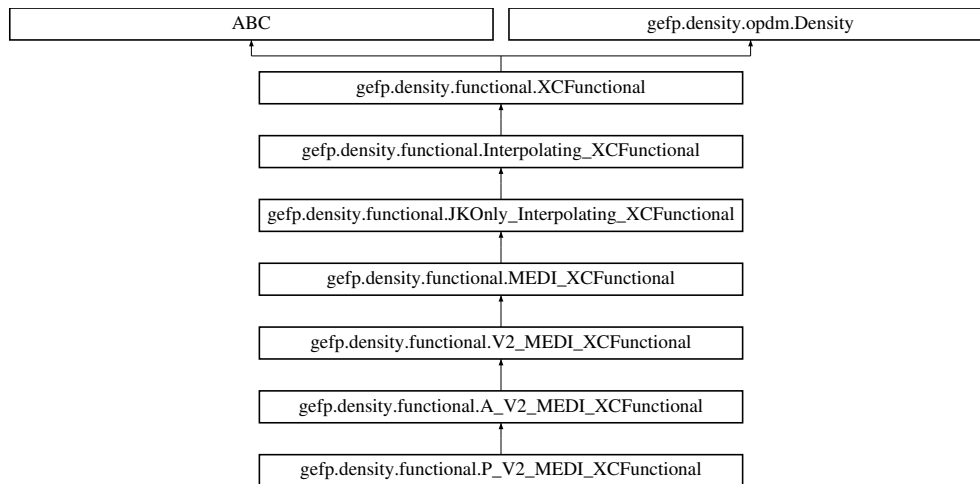
- 't' - t parameter in Gegenbauer Series. Parameter t to be between 0.0 and 1.0. The larger t, the more terms should be included (higher kmax).

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

- `gefp/gefp/density/functional.py`

## 18.156 gefp.density.functional.V2\_MEDI\_XCFunctional Class Reference

Inheritance diagram for `gefp.density.functional.V2_MEDI_XCFunctional`:



## Public Member Functions

- `def __init__ (self, coeff, kmax)`
- `def compute_ak (self, k, t)`
- `def compute_t (self, n=None, c=None)`
- `def compute_ak_derivative_t (self, k, t)`

## Static Public Attributes

- `float parameter_A = 100.0`
- `float parameter_L = -0.4`

## Additional Inherited Members

### 18.156.1 Detailed Description

\
 Version 2 of MEDI functional. It is a proper functional of density matrix. Also, it can be represented explicitly in terms of density matrix.

Functional coefficients:

- 't' - t parameter in Gegenbauer Series. Parameter t to be between 0.0 and 1.0. The larger t, the more terms should be included (higher kmax).

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

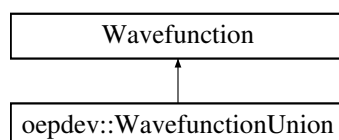
- `gefp/gefp/density/functional.py`

## 18.157 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.*

- [WavefunctionUnion](#) (SharedMolecule dimer, SharedBasisSet [primary](#), SharedBasisSet auxiliary\_df, SharedBasisSet guess, SharedBasisSet primary\_1, SharedBasisSet primary\_2, SharedBasisSet auxiliary\_1, SharedBasisSet auxiliary\_2, SharedBasisSet auxiliary\_df\_1, SharedBasisSet auxiliary\_df\_2, SharedBasisSet intermediate\_1, SharedBasisSet intermediate\_2, SharedBasisSet guess\_1, SharedBasisSet guess\_2, SharedWavefunction wfn\_1, SharedWavefunction wfn\_2, 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)*

- void [clear\\_dpd](#) ()

*Close the DPD instance.*

- 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\\_alpha](#) (int n) const
- Get the number of the alpha electrons of the \*n\*th fragment.*

  - int [L\\_beta](#) (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.*

  - SharedBasisSet [L\\_guess](#) (int n) const
- Get the guess 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 &nocpi, 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< SharedBasisSet > [l\\_guess\\_](#)  
*List of guess basis functions per molecule.*
- std::vector< SharedWavefunction > [l\\_wfn\\_](#)  
*List of original isolated wavefunctions (electrons unrelaxed)*
- std::vector< std::string > [l\\_name\\_](#)  
*List of names of isolated wavefunctions.*
- std::vector< int > [l\\_nbf\\_](#)  
*List of basis function numbers per molecule.*
- std::vector< int > [l\\_nmo\\_](#)  
*List of numbers of molecular orbitals (MO's) per molecule.*
- std::vector< int > [l\\_nso\\_](#)  
*List of numbers of SO's per molecule.*



- `std::vector< int > l_ndocc_`  
*List of numbers of doubly occupied orbitals per molecule.*
- `std::vector< int > l_nvir_`  
*List of numbers of virtual orbitals per molecule.*
- `std::vector< int > l_noffs_ao_`  
*List of basis set offsets per molecule.*
- `std::vector< double > l_energy_`  
*List of energies of isolated wavefunctions.*
- `std::vector< double > l_efzc_`  
*List of frozen-core energies per isolated wavefunction.*
- `std::vector< bool > l_density_fitted_`  
*List of information per wfn whether it was obtained using DF or not.*
- `std::vector< int > l_nalpha_`  
*List of numbers of alpha electrons per isolated wavefunction.*
- `std::vector< int > l_nbeta_`  
*List of numbers of beta electrons per isolated wavefunction.*
- `std::vector< int > l_nfrzc_`  
*List of numbers of frozen-core orbitals per isolated molecule.*
- `std::vector< SharedLocalizer > l_localizer_`  
*List of orbital localizers.*
- `std::vector< std::map< const std::string, SharedMOSpace > > l_mospace_`  
*List of dictionaries of MO spaces.*
- `std::shared_ptr< psi::OEProp > oeprop_`  
*One-Electron Property.*

### 18.157.1 Detailed Description

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
- 
1. Performs Hadamard sums on `H_`, `Fa_`, `Da_`, `Ca_` and `S_` based on uncoupled wavefunctions.
  2. 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)
- `frzcp_` (Number of frozen core per irrep)
- `frzvp_` (Number of frozen virtuals per irrep)
- `nalphapi_` (Number of alpha electrons per irrep)
- `nbetapi_` (Number of beta electrons per irrep)
- `nsopi_` (Number of so per irrep)
- `nmopi_` (Number of mo per irrep)
- `nso_` (Total number of SOs)
- `nmo_` (Total number of MOs)
- `nirrep_` (Number of irreps; must be equal to 1 due to symmetry reasons)
- `same_a_b_dens_` and `same_a_b_orbs_` The rest is altered so that the Wavefunction parameters reflect a cluster of non-interacting (uncoupled, isolated, unrelaxed) molecular electron densities.

## 18.157.2 Constructor & Destructor Documentation

### 18.157.2.1 WavefunctionUnion() [1/2]

```
oepdev::WavefunctionUnion::WavefunctionUnion (
    SharedWavefunction ref_wfn,
    Options & options )
```

Provide wavefunction with molecule containing at least 2 fragments.

#### Parameters

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

### 18.157.2.2 WavefunctionUnion() [2/2]

```
oepdev::WavefunctionUnion::WavefunctionUnion (
    SharedMolecule dimer,
    SharedBasisSet primary,
    SharedBasisSet auxiliary_df,
    SharedBasisSet guess,
    SharedBasisSet primary_1,
    SharedBasisSet primary_2,
    SharedBasisSet auxiliary_1,
    SharedBasisSet auxiliary_2,
    SharedBasisSet auxiliary_df_1,
    SharedBasisSet auxiliary_df_2,
    SharedBasisSet intermediate_1,
    SharedBasisSet intermediate_2,
    SharedBasisSet guess_1,
    SharedBasisSet guess_2,
    SharedWavefunction wfn_1,
    SharedWavefunction wfn_2,
    Options & options )
```

Provide molecule dimer and all the required monomer basis sets and wavefunctions.

#### Parameters

<i>dimer</i>	- molecule object
<i>primary</i>	- basis set object: dimer (primary)
<i>auxiliary_df</i>	- basis set object: dimer (DF SCF)
<i>guess</i>	- basis set object: dimer (guess)

**Parameters**

<i>primary_1</i>	- basis set object for 1st monomer
<i>primary_2</i>	- basis set object for 2nd monomer
<i>auxiliary_1</i>	- basis set object for 1st monomer
<i>auxiliary_2</i>	- basis set object for 2nd monomer
<i>auxiliary_df_1</i>	- basis set object for 1st monomer
<i>auxiliary_df_2</i>	- basis set object for 2nd monomer
<i>intermediate_1</i>	- basis set object for 1st monomer
<i>intermediate_2</i>	- basis set object for 2nd monomer
<i>guess_1</i>	- basis set object for 1st monomer
<i>guess_2</i>	- basis set object for 2nd monomer
<i>wfn_1</i>	- unperturbed wavefunction object
<i>wfn_2</i>	- unperturbed wavefunction object
<i>options</i>	- Psi4 options

**18.157.3 Member Function Documentation****18.157.3.1 Ca\_subset()**

```
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

**18.157.3.2 Cb\_subset()**

```
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

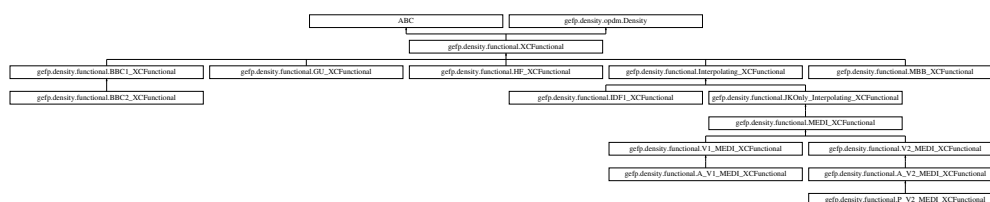
the matrix in Pitzer order in the desired basis

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

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

## 18.158 `gefp.density.functional.XCFunctional` Class Reference

Inheritance diagram for `gefp.density.functional.XCFunctional`:



### Public Member Functions

- `def __init__ (self, jk=None, wfn=None, ints=None)`
- `def set_wfn (self, wfn)`
- `def set_jk (self, jk)`
- `def set_ints (self, ints)`
- `def create (cls, name=default, kwargs)`
- `def abbr (self)`
- `def energy_D (self, x, mode)`
- `def energy_P (self, x)`
- `def energy_pc (self, x)`
- `def gradient_D (self, x)`
- `def gradient_P (self, x)`
- `def gradient_P_approximate (self, x)`
- `def gradient_P_numerical (self, x)`
- `def gradient_nc (self, x)`
- `def gradient_pc (self, x)`

## Static Public Member Functions

- def **name** ()
- def **fij** (n)
- def **fij\_1** (n, m)

## Static Public Attributes

- string **default** = 'hf'

### 18.158.1 Detailed Description

\

The Exchange-Correlation DMFT functional.

### 18.158.2 Member Function Documentation

#### 18.158.2.1 create()

```
def gefp.density.functional.XCFunctional.create (
    cls,
    name = default,
    kwargs )
```

\

Create a density matrix exchange-correlation functional.

Available functionals:	Sets:	Analytic Derivatives:
o 'HF' - the Hartree-Fock functional (default)	D, NC	Yes
o 'MBB' - the Muller-Buijse-Baerends functional	P	Yes
o 'GU' - the Goedecker-Urmigar functional	P	Approximate
o 'BBC1' - the BBC1 functional	P	No
o 'BBC2' - the BBC2 functional	P	No
o 'MEDI' - the monotonous exponential decay of interpolates between MBB and MBB with zero exchange.	P	No
o 'OEDI' - the oscillatory exponential decay of interpolates between MBB and MBB with zero exchange.	P	No
o 'IDF1' - interpolating density matrix functional ???	P	No

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

- gefp/gefp/density/functional.py

# Chapter 19

## File Documentation

### 19.1 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 ( $3 * (OEPDEV\_MAX\_AM + 1) * (OEPDEV\_MAX\_AM + 1) * OEPDEV\_N\_MAX\_AM$ )*
- #define `OEPDEV_AU_KcalPerMole` 627.509  
*Energy converters.*
- #define `OEPDEV_AU_CMRec` 219474.63
- #define `OEPDEV_AU_EV` 27.21138

### 19.2 include/oepdev\_options.h File Reference

#### Namespaces

- `psi`  
*Psi4 package namespace.*

## Functions

- PSI.API int [psi::read\\_options](#) (std::string name, Options &options)

*Options for the OEPEDev plugin.*

## 19.3 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"
#include <pybind11/pybind11.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 **SharedGEFFFactory** = std::shared\_ptr< [oepdev::GenEffParFactory](#) >
- using **SharedGEFFParameters** = std::shared\_ptr< [oepdev::GenEffPar](#) >

## Functions

- void **psi::export\_dmtf** (py::module &)
- void **psi::export\_cphf** (py::module &)
- void **psi::export\_solver** (py::module &)
- void **psi::export\_util** (py::module &)
- void **psi::export\_oep** (py::module &)
- void **psi::export\_gefp** (py::module &)
- PSI.API SharedWavefunction [psi::oepdev](#) (SharedWavefunction ref\_wfn, Options &options)



*Main routine of the OEPLDev plugin.*

- `psi::PYBIND11_MODULE` (oepdev, m)

## 19.4 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`  
*OEPLDev module namespace.*

### Typedefs

- using `psi::SharedBasisSet` = `std::shared_ptr< BasisSet >`

## 19.5 oepdev/lib3d/esp.h File Reference

```
#include "space3d.h"
```

### Classes

- class `oepdev::ESPSolver`  
*Charges from Electrostatic Potential (ESP). A solver-type class.*

## Namespaces

- [oepdev](#)

*OEPDev module namespace.*

## Typedefs

- using [oepdev::SharedField3D](#) = std::shared\_ptr< [oepdev::Field3D](#) >

## 19.6 oepdev/libgefp/gefp.h File Reference

```
#include <vector>
#include <string>
#include <random>
#include <cmath>
#include <map>
#include "psi4/libpsi4util/PsiOutStream.h"
#include "psi4/libpsi4util/process.h"
#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::EFP2\\_GEFactory](#)  
*EFP2 GEFP Factory.*
- class [oepdev::OEP\\_EFP2\\_GEFactory](#)  
*OEP-EFP2 GEFP Factory.*
- 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.*

## 19.7 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  $(a|O(2)|b)$  where  $O(2) = 1/r^{12}$ .*
- class [oepdev::ERI\\_2\\_2](#)  
*4-centre ERI of the form  $(ab|O(2)|cd)$  where  $O(2) = 1/r^{12}$ .*
- class [oepdev::ERI\\_3\\_1](#)  
*4-centre ERI of the form  $(abc|O(2)|d)$  where  $O(2) = 1/r^{12}$ .*

## Namespaces

- [oepdev](#)  
*OEPEv module namespace.*

## 19.8 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.*

## 19.9 oepdev/liboep/oep.h File Reference

```
#include <cstdio>
#include <string>
#include <vector>
#include <map>
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsi4util/PsiOutStream.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/local.h"
#include "../libutil/cis.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](#)

*OEPEDev 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 >
- using **oepdev::SharedLocalizer** = std::shared\_ptr< Localizer >
- using **oepdev::SharedCISData** = std::shared\_ptr< CISData >

## 19.10 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.*

## 19.11 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.*

## 19.12 oepdev/libpsi/potential.h File Reference

```
#include <vector>
#include "psi4/psi4-dec.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.*

## 19.13 oepdev/libsolver/solver.h File Reference

```
#include <cstdio>
#include <string>
#include <map>
#include "psi4/psi4-dec.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libmints/potential.h"
#include "psi4/libmints/integral.h"
#include "psi4/libpsi4util/PsiOutStream.h"
#include "../libutil/wavefunction_union.h"
#include "../libutil/integrals_iter.h"
#include "../libpsi/integral.h"
#include "../liboep/oep.h"
#include "../../include/oepdev_files.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.*



- class [oepdev::EETCouplingSolver](#)

*Compute the EET coupling energy between unperturbed wavefunctions.*

## Namespaces

- [oepdev](#)

*OEPEv module namespace.*

## Typedefs

- using **oepdev::SharedWavefunctionUnion** = std::shared\_ptr< WavefunctionUnion >
- using **oepdev::SharedOEPotential** = std::shared\_ptr< OEPotential >

## 19.14 oepdev/libsolver/ti\_data.h File Reference

```
#include <cstdio>
#include <string>
#include <map>
#include "../lib3d/dmtp.h"
```

## Classes

- class [oepdev::TIData](#)

*Transfer Integral EET Data.*

## Namespaces

- [oepdev](#)

*OEPEv module namespace.*

## Typedefs

- using **oepdev::SharedDMTPConvergence** = std::shared\_ptr< [oepdev::MultipoleConvergence](#) >

## 19.15 oepdev/libtest/test.h File Reference

```
#include <vector>
#include "psi4/psi4-dec.h"
```

```
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libpsi4util/PsiOutStream.h"
#include "psi4/libmints/integral.h"
#include "psi4/libmints/matrix.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"
#include "../../include/oepdev_files.h"
```

## Classes

- class [oepdev::test::Test](#)

*Manages test routines.*

## Namespaces

- [oepdev](#)

*OEPEv module namespace.*

## 19.16 oepdev/libutil/basis\_rotation.h File Reference

```
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/basisset.h"
```

## Namespaces

- [psi](#)

*Psi4 package namespace.*

- [oepdev](#)

*OEPEv module namespace.*

## Functions

### Rotation of AO Space

### 19.16.1 Theory

The objective is to find the formulae for rotation matrices of the AO spaces as functions of the Cartesian 3 x 3 rotation matrices. It is obvious that p-type functions transform as a usual Cartesian vectors. However, higher angular momentum functions transform in a more complex way.

#### Problem

Define a vectorized AO space  $M$  of rank  $r > 1$  that is constructed from unique tensor components of fully symmetric  $r$ -th rank AO tensor populated in standard order,

$$M_{\{ab\dots k\}} = M_{ab\dots k} \quad \text{for } a \leq b \leq \dots \leq k$$

Given a general rotation of Cartesian tensors

$$M_{ab\dots k} = \sum_{a'b'\dots k'} M_{a'b'\dots k'} r_{a'a} r_{b'b} \dots r_{k'k}$$

find closed expressions for the rotation matrix in reduced composite AO space obeying

$$M_{[ab\dots k]} = \sum_{\{a'b'\dots k'\}} M_{\{a'b'\dots k'\}} R_{\{a'b'\dots k'\}, [ab\dots k]}$$

In the derivations below the following identity of first-order partitioning will be of use:

$$\sum_{ab} M_{ab} \hat{s}_{ab} = \sum_{\{ab\}} M_{\{ab\}} (\hat{s}_{ab} + \Delta_{ab} \hat{s}_{ba})$$

where

$$\Delta_{ab} \equiv 1 - \delta_{ab}$$

and the operator  $s$  of rank  $r$  acts as follows

$$s_{a'b'\dots k'}^{ab\dots k} \equiv \hat{s}_{a'b'\dots k'} \underbrace{\mathbf{r} \otimes \mathbf{r} \otimes \dots \otimes \mathbf{r}}_r = r_{a'a} r_{b'b} \dots r_{k'k}$$

#### Rotation of 6D functions

The rotation of the 6D vector full tensor AO space of rank 2 and dimensions (3,3) is given by

$$M_{ab} = \sum_{a'b'} M_{a'b'} r_{a'a} r_{b'b}$$

Applying the identity of first-order partitioning directly leads to the formula for a reduced 6D tensor rotation of rank 1 and dimension (6),

$$M_{[ab]} = \sum_{\{a'b'\}} M_{\{a'b'\}} R_{\{a'b'\}, [ab]}$$

where the 6 x 6 rotation matrix is given by

$$R_{\{a'b'\}, [ab]} = r_{a'a} r_{b'b} + \Delta_{a'b'} r_{b'a} r_{a'b}$$

### Rotation of 10F functions

The rotation of the 10D vector full tensor AO space of rank 3 and dimensions (3,3,3) is given by

$$M_{abc} = \sum_{a'b'c'} M_{a'b'c'} r_{a'b} r_{b'b} r_{c'c}$$

First of all, notice that one can perform the following partitioning

$$\sum_a \sum_{b \neq a} \sum_{c \neq b \neq a} M_{abc} \hat{s}_{abc} = \sum_{\{abc\}} M_{\{abc\}} (\hat{s}_{abc} + \hat{s}_{acb} + \hat{s}_{bac} + \hat{s}_{bca} + \hat{s}_{cab} + \hat{s}_{cba})$$

Then, perform a partitioning of the triple sum,

$$\begin{aligned} \sum_{abc} M_{abc} \hat{s}_{abc} &= \sum_a \sum_{b \neq a} \sum_{c \neq b \neq a} M_{abc} \hat{s}_{abc} \\ &+ \sum_a \sum_{b \geq a} M_{abb} \hat{s}_{abb} + \sum_a \sum_{b < a} M_{abb} \hat{s}_{abb} \\ &+ \sum_a \sum_{b > a} M_{aba} \hat{s}_{aba} + \sum_a \sum_{b < a} M_{aba} \hat{s}_{aba} \\ &+ \sum_a \sum_{b > a} M_{bba} \hat{s}_{bba} + \sum_a \sum_{b < a} M_{bba} \hat{s}_{bba} \end{aligned}$$

Using the first-order partitioning theorem and interchanging the dummy indices one finds that

$$M_{[abc]} = \sum_{\{a'b'c'\}} M_{\{a'b'c'\}} R_{\{a'b'c'\},[abc]}$$

where the 10 x 10 rotation matrix is given by

$$\begin{aligned} R_{\{a'b'c'\},[abc]} &= \delta_{b'c'} \left( s_{a'b'b'}^{abc} + \Delta_{a'b'} \left\{ s_{b'a'b'}^{abc} + s_{b'b'a'}^{abc} \right\} \right) \\ &+ \delta_{a'b'} \Delta_{b'c'} \left( s_{c'a'a'}^{abc} + s_{a'c'a'}^{abc} + s_{a'a'c'}^{abc} \right) \\ &+ \Delta_{a'b'} \Delta_{b'c'} \left( s_{a'b'c}^{abc} + s_{a'c'b'}^{abc} + s_{b'a'c'}^{abc} + s_{b'c'a'}^{abc} + s_{c'a'b'}^{abc} + s_{c'b'a'}^{abc} \right) \end{aligned}$$

where

$$s_{a'b'c'}^{abc} \equiv \hat{s}_{a'b'c'} \mathbf{r} \otimes \mathbf{r} \otimes \mathbf{r} = r_{a'a} r_{b'b} r_{c'c}$$

- psi::SharedMatrix [oepdev::r6](#) (psi::SharedMatrix r)  
Compute the 6 x 6 rotation matrix of the 6D orbitals.
- void [oepdev::populate](#) (double \*\*R, double \*\*r, std::vector< int > idx\_am, const int &nam)  
Compute the 6 x 6 rotation matrix of the 6D orbitals.
- psi::SharedMatrix [oepdev::ao\\_rotation\\_matrix](#) (psi::SharedMatrix r, psi::SharedBasisSet b)  
Compute the full rotation matrix of AO orbital space.

## 19.17 oepdev/libutil/cis.h File Reference

```
#include <string>
#include <utility>
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libpsio/psio.hpp"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/molecule.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libtrans/integraltransform.h"
#include "psi4/libtrans/mospace.h"
#include "psi4/libdpd/dpd.h"
#include "psi4/libfock/jk.h"
#include "../lib3d/dmtp.h"
#include "davidson_liu.h"
```

### Classes

- struct [oepdev::CISData](#)

*Container to handle the CIS wavefunction parameters.*

- class [oepdev::CISComputer](#)

*CISComputer.*

- class [oepdev::R\\_CISComputer](#)
- class [oepdev::U\\_CISComputer](#)
- class [oepdev::R\\_CISComputer\\_Explicit](#)
- class [oepdev::R\\_CISComputer\\_DL](#)

*CIS Computer with RHF reference: Davidson-Liu Solver.*

- class [oepdev::R\\_CISComputer\\_Direct](#)
- class [oepdev::U\\_CISComputer\\_Explicit](#)
- class [oepdev::U\\_CISComputer\\_DL](#)

*CIS Computer with UHF reference: Davidson-Liu Solver.*

### Namespaces

- [oepdev](#)

*OEPEDev module namespace.*

## Typedefs

- using **oepdev::SharedMolecule** = std::shared\_ptr< psi::Molecule >
- using **oepdev::SharedMOSpace** = std::shared\_ptr< psi::MOSpace >
- using **oepdev::SharedMOSpaceVector** = std::vector< std::shared\_ptr< psi::MOSpace > >
- using **oepdev::SharedIntegralTransform** = std::shared\_ptr< psi::IntegralTransform >

## 19.18 oepdev/libutil/davidson\_liu.h File Reference

```
#include "psi4/liboptions/liboptions.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/matrix.h"
#include "gram_schmidt.h"
```

### Classes

- class [oepdev::DavidsonLiu](#)  
*Davidson-Liu diagonalization method.*

### Namespaces

- [oepdev](#)  
*OEPEv module namespace.*

## 19.19 oepdev/libutil/diis.h File Reference

```
#include <cstdio>
#include <string>
#include <vector>
#include "psi4/libciomr/libciomr.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libqt/qt.h"
#include "psi4/libpsi4util/PsiOutStream.h"
```

### Classes

- class [oepdev::DIISManager](#)  
*DIIS manager.*

## Namespaces

- [oepdev](#)

*OEPEv module namespace.*

## 19.20 oepdev/libutil/gram\_schmidt.h File Reference

```
#include "psi4/libmints/vector.h"
```

## Classes

- class [oepdev::GramSchmidt](#)

*Gram-Schmidt orthogonalization method.*

## Namespaces

- [oepdev](#)

*OEPEv module namespace.*

## 19.21 oepdev/libutil/integrals\_iter.h File Reference

```
#include <cstdio>
#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](#)

*OEPEv 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.*

## 19.22 oepdev/libutil/kabsch\_superimposer.h File Reference

```
#include <string>
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/molecule.h"
```

## Classes

- class [oepdev::KabschSuperimposer](#)

*Compute the Cartesian rotation matrix between two structures.*

## Namespaces

- [oepdev](#)

*OEPEv module namespace.*

## 19.23 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::RHPerturbed](#)  
*RHF theory under electrostatic perturbation.*

## Namespaces

- [oepdev](#)  
*OEPEv module namespace.*

## 19.24 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 optimum unitary matrix of quadratic matrix equation.*
- class [oepdev::UnitaryOptimizer\\_4\\_2](#)  
*Find the optimum 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)

## 19.25 oeplibutil/util.h File Reference

```
#include <cstdio>
#include <string>
#include <cmath>
#include <map>
#include <cassert>
#include "psi4/psi4-dec.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/oeoprop.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

- [oeplib](#)  
*OEPLib module namespace.*

## Typedefs

- using **oeplib::SharedSuperFunctional** = `std::shared_ptr< SuperFunctional >`

## Functions

- PSI.API 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);.*
- PSI.API std::shared\_ptr< SuperFunctional > [oepdev::create\\_superfunctional](#) (std::string name, Options &options)  
*Set up DFT functional.*
- PSI.API std::shared\_ptr< Molecule > [oepdev::extract\\_monomer](#) (std::shared\_ptr< const Molecule > molecule\_dimer, int id)  
*Extract molecule from dimer.*
- PSI.API double [oepdev::compute\\_distance](#) (psi::SharedVector v1, psi::SharedVector v2)  
*Compute distance between two points in nD space.*
- PSI.API std::shared\_ptr< Wavefunction > [oepdev::solve\\_scf](#) (std::shared\_ptr< Molecule > molecule, std::shared\_ptr< BasisSet > primary, std::shared\_ptr< BasisSet > auxiliary, std::shared\_ptr< BasisSet > guess, std::shared\_ptr< SuperFunctional > functional, Options &options, std::shared\_ptr< PSIO > psio, bool compute\_mints=false)  
*Solve RHF-SCF equations for a given molecule in a given basis set.*
- PSI.API std::shared\_ptr< Wavefunction > [oepdev::solve\\_scf\\_sad](#) (std::shared\_ptr< Molecule > molecule, std::shared\_ptr< BasisSet > primary, std::shared\_ptr< BasisSet > auxiliary, std::vector< std::shared\_ptr< BasisSet >> sad, std::vector< std::shared\_ptr< BasisSet >> sad\_fit, std::shared\_ptr< SuperFunctional > functional, Options &options, std::shared\_ptr< PSIO > psio, bool compute\_mints=false)  
*Solve RHF-SCF equations for a given molecule in a given basis set.*
- PSI.API double [oepdev::average\\_moment](#) (std::shared\_ptr< psi::Vector > moment)  
*Compute the scalar magnitude of multipole moment.*
- PSI.API std::vector< std::shared\_ptr< psi::Matrix >> [oepdev::calculate\\_JK](#) (std::shared\_ptr< psi::Wavefunction > wfn, std::shared\_ptr< psi::Matrix > C)  
*Compute the Coulomb and exchange integral matrices in MO basis.*
- PSI.API std::vector< std::shared\_ptr< psi::Matrix >> [oepdev::calculate\\_JK\\_r](#) (std::shared\_ptr< psi::Wavefunction > wfn, std::shared\_ptr< psi::IntegralTransform > tr, std::shared\_ptr< psi::Matrix > Dij)  
*Compute the Coulomb and exchange integral matrices in MO basis.*
- PSI.API std::shared\_ptr< psi::Matrix > [oepdev::calculate\\_der\\_D](#) (std::shared\_ptr< psi::Wavefunction > wfn, std::shared\_ptr< psi::IntegralTransform > tr, std::shared\_ptr< psi::Matrix > C, std::vector< std::shared\_ptr< psi::Matrix >> A)  
*Compute the derivative of exchange-correlation energy wrt the density matrix in MO-A basis.*
- PSI.API double [oepdev::calculate\\_e\\_xc](#) (std::shared\_ptr< psi::Wavefunction > wfn, std::shared\_ptr< psi::IntegralTransform > tr, std::shared\_ptr< psi::Matrix > f, std::shared\_ptr< psi::Matrix > C)  
*Compute the exchange-correlation energy from ERI in MO-SCF basis.*

- PSI.API std::shared\_ptr< psi::Matrix > [oepdev::matrix\\_power\\_derivative](#) (std::shared\_ptr< psi::Matrix > A, double g, double step)

*Compute the contracted derivative of power of a square and symmetric matrix.*

- std::shared\_ptr< psi::Matrix > [oepdev::calculate\\_DFI\\_Vel](#) (std::shared\_ptr< psi::IntegralFactory > f\_aabb, std::shared\_ptr< psi::IntegralFactory > f\_abab, std::shared\_ptr< psi::Matrix > d.b)

*Compute the Effective DFI Potential Matrix Due To Electrons.*

- PSI.API std::shared\_ptr< psi::Matrix > [oepdev::calculate\\_DFI\\_Vel\\_JK](#) (std::shared\_ptr< psi::IntegralFactory > f\_aabb, std::shared\_ptr< psi::IntegralFactory > f\_abab, std::shared\_ptr< psi::Matrix > d.b)

*Compute the Effective DFI Coulomb+Exchange Potential Matrix Due To Electrons.*

- PSI.API std::shared\_ptr< psi::Matrix > [oepdev::calculate\\_DFI\\_Vel\\_J](#) (std::shared\_ptr< psi::IntegralFactory > f\_aabb, std::shared\_ptr< psi::Matrix > d.b)

*Compute the Effective DFI Coulomb Potential Matrix Due To Electrons.*

## 19.26 oepdev/libutil/wavefunction\_union.h File Reference

```
#include <cstdio>
#include <string>
#include <map>
#include "psi4/psi4-dec.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 20

## Example Documentation

### 20.1 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);  
};
```

### 20.2 example\_davidson\_liu.cc

This example is a trivial demo to use `oepdev::DavidsonLiu` in order to diagonalize a real, symmetric matrix **H**, stored in a `psi::SharedMatrix H`. This can help you to construct more complicated classes that need to solve eigenpairs for very large, sparse matrices such as CI Hamiltonians.

#### Note

This example might need compile properly (it's only a draft). Debug if necessary.

```
// Define a class that inherits from DavidsonLiu  
class Diagonalize : public DavidsonLiu {  
  
public:
```

```

Diagonalize(psi::SharedMatrix H, psi::Options& opt, int M);
virtual void ~Diagonalize() {};

protected:
    // Desired number of roots to be found
    int M_;
    // Matrix to be diagonalized (explicitly stored)
    psi::SharedMatrix matrix_;

    // Implementation of pure methods must be declared
    void davidson_liu_compute_diagonal_hamiltonian();
    void davidson_liu_compute_sigma();
};

Diagonalize::Diagonalize(psi::SharedMatrix H, psi::Options& opt, int M) :
    DavidsonLiu(opt) , matrix_(nullptr), M_(M)
{
    matrix_ = std::make_shared<psi::Matrix>(H);
    int N = H->ncol();
    int L = M_;

    // Must be run in order to allocate memory
    this->davidson_liu_initialize(N, L, M_);
}

// Implementation of pure methods
void Diagonalize::davidson_liu_compute_diagonal_hamiltonian() {
    for (int i=0; i<this->matrix_->ncol(); ++i) {
        double v = matrix_->get(i, i);
        this->H_diag_davidson_liu->set(i, v);
    }
}

void Diagonalize::davidson_liu_compute_sigma() {
    for (int k=this->davidson_liu_n_sigma_computed; k<this->L.davidson_liu; ++k) {
        psi::SharedVector Sigma = std::make_shared<psi::Vector>("", this->N.davidson_liu);
        Sigma->gemv(false, 1.0, *this->matrix_, *Sigma, 0.0);
        this->sigma_vectors_davidson_liu.push_back(Sigma);
    }
}

// Testing function
void example_davidson_liu(psi::SharedMatrix H, int M, psi::Options& opt){

    // Construct the solver object
    Diagonalize solver(H, opt, M);

    // Find *M* lowest eigenpairs of a given matrix **H**
    solver.run_davidson_liu();
};

```

## 20.3 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()];
    }
}
}

```

## 20.4 example\_scf\_perturb.cc

Perturb HF Hamiltonian with external electrostatic potential. This is an example of how to use the `oepdev::RHFPerturbed` 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::RHFPerturbed> scf = std::make_shared<oepdev::RHFPerturbed>(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

- `__init__`
  - `gefp::core::utilities::UnitaryOptimizer`, [368](#)
  - `gefp::core::utilities::UnitaryOptimizer_4_2`, [376](#)
  - `gefp::math::matrix::Superimposer`, [339](#)
- `_calculate_DFI_Vel`
  - The OEPDev Utilities, [76](#)
- `AllAOIntegralsIterator_2`
  - `oepdev::AllAOIntegralsIterator_2`, [107](#)
- `AllAOIntegralsIterator_4`
  - `oepdev::AllAOIntegralsIterator_4`, [109](#)
- `AllAOShellCombinationsIterator_2`
  - `oepdev::AllAOShellCombinationsIterator_2`, [111](#), [112](#)
- `AllAOShellCombinationsIterator_4`
  - `oepdev::AllAOShellCombinationsIterator_4`, [114](#), [115](#)
- `allocate`
  - `oepdev::GenEffPar`, [226](#)
- `ao_iterator`
  - `oepdev::ShellCombinationsIterator`, [332](#)
- `ao_rotation_matrix`
  - The OEPDev Utilities, [77](#)
- `atomic_charges`
  - `gefp::density::population`, [89](#)
- `average_moment`
  - The OEPDev Utilities, [77](#)
- `basisset`
  - `oepdev::GenEffPar`, [227](#)
- `build`
  - `oepdev::AOIntegralsIterator`, [117](#)
  - `oepdev::CISComputer`, [130](#)
  - `oepdev::DMTPole`, [176](#)
  - `oepdev::Field3D`, [214](#)
  - `oepdev::GenEffParFactory`, [236](#)
  - `oepdev::GeneralizedDensityFit`, [239](#), [240](#)
  - `oepdev::OEPDevSolver`, [283](#)
  - `oepdev::OEPotential`, [288](#)
  - `oepdev::Points3DIterator`, [297](#), [298](#)
  - `oepdev::PointsCollection3D`, [300](#), [301](#)
  - `oepdev::ShellCombinationsIterator`, [333](#)
- `CPHF`
  - `oepdev::CPHF`, [140](#)
- `Ca_subset`
  - `oepdev::WavefunctionUnion`, [388](#)
- `calculate_DFI_Vel_JK`
  - The OEPDev Utilities, [78](#)
- `calculate_DFI_Vel_J`
  - The OEPDev Utilities, [78](#)
- `calculate_JK_r`
  - The OEPDev Utilities, [80](#)
- `calculate_JK`
  - The OEPDev Utilities, [79](#)
- `calculate_der_D`
  - The OEPDev Utilities, [77](#)
- `calculate_e_xc`
  - The OEPDev Utilities, [79](#)
- `Cb_subset`
  - `oepdev::WavefunctionUnion`, [388](#)
- `compute`
  - `gefp::density::partitioning::DensityDecomposition`, [153](#)
  - `oepdev::DIISManager`, [160](#)
  - `oepdev::DMTPole`, [176](#)
  - `oepdev::DoubleGeneralizedDensityFit`, [184](#)
  - `oepdev::GeneralizedDensityFit`, [240](#)
  - `oepdev::KabschSuperimposer`, [258](#)
  - `oepdev::MultipoleConvergence`, [267](#)
  - `oepdev::SingleGeneralizedDensityFit`, [337](#)
  - `oepdev::TwoBodyAOInt`, [354](#)
- `compute_benchmark`
  - `oepdev::ChargeTransferEnergySolver`, [126](#)
  - `oepdev::EETCouplingSolver`, [192](#)
  - `oepdev::ElectrostaticEnergySolver`, [199](#)
  - `oepdev::OEPDevSolver`, [283](#)
  - `oepdev::RepulsionEnergySolver`, [324](#)
- `compute_density_matrix`
  - `oepdev::GenEffPar`, [227](#), [228](#)

- compute\_distance
  - The OEPDev Utilities, [80](#)
- compute\_energy
  - oepdev::GenEffFrag, [220](#)
- compute\_many\_body\_energy
  - oepdev::GenEffFrag, [220](#)
- compute\_oep\_based
  - oepdev::ChargeTransferEnergySolver, [126](#)
  - oepdev::EETCouplingSolver, [192](#)
  - oepdev::ElectrostaticEnergySolver, [199](#)
  - oepdev::OEPDevSolver, [284](#)
  - oepdev::RepulsionEnergySolver, [324](#)
- compute\_shell
  - oepdev::AllAOShellCombinationsIterator\_2, [112](#)
  - oepdev::AllAOShellCombinationsIterator\_4, [115](#)
  - oepdev::ShellCombinationsIterator, [334](#)
  - oepdev::TwoElectronInt, [356](#)
- ConvergenceLevel
  - oepdev::MultipoleConvergence, [266](#)
- coupling\_direct
  - oepdev::TIDData, [347](#)
- coupling\_direct\_coul
  - oepdev::TIDData, [347](#)
- coupling\_direct\_exch
  - oepdev::TIDData, [347](#)
- coupling\_indirect
  - oepdev::TIDData, [348](#)
- coupling\_indirect\_ti2
  - oepdev::TIDData, [348](#)
- coupling\_indirect\_ti3
  - oepdev::TIDData, [348](#)
- coupling\_total
  - oepdev::TIDData, [348](#)
- coupling\_trcamm
  - oepdev::TIDData, [349](#)
- create
  - gefp::density::dmft::DMFT, [163](#)
  - gefp::density::functional::XCFunctional, [390](#)
- create\_superfunctional
  - The OEPDev Utilities, [81](#)
- d\_N\_n1\_n2
  - The Integral Package Library, [62](#)
- DIISManager
  - oepdev::DIISManager, [159](#)
- DMTPole
  - oepdev::DMTPole, [175](#)
- deformation\_density
  - gefp::density::partitioning::DensityDecomposition, [153](#)
- determine\_dmt\_pole\_convergence\_level
  - oepdev::DMTPole, [176](#)
- dmt\_pole
  - oepdev::GenEffPar, [228](#)
- dipole
  - oepdev::GenEffPar, [229](#)
- ESPSolver
  - oepdev::ESPSolver, [208](#), [209](#)
- empty
  - oepdev::DMTPole, [177](#)
- energy
  - oepdev::DMTPole, [177](#)
  - oepdev::GenEffFrag, [220](#)
- extract\_monomer
  - The OEPDev Utilities, [81](#)
- field
  - oepdev::DMTPole, [178](#)
- Field3D
  - oepdev::Field3D, [213](#)
- gefp.basis.optimize, [87](#)
- gefp.basis.optimize.DFBasis, [154](#)
- gefp.basis.optimize.DFBasisOptimizer, [155](#)
- gefp.basis.optimize.OEP\_CT, [272](#)
- gefp.basis.optimize.OEP\_FockLike, [273](#)
- gefp.basis.optimize.OEP\_Pauli, [274](#)
- gefp.basis.optimize.OEP, [271](#)
- gefp.core.driver.Entry, [201](#)
- gefp.core.driver.PadeApproximant\_2D, [293](#)
- gefp.core.driver.UniversalSurface, [378](#)
- gefp.core.utilities.UnitaryOptimizer, [367](#)
- gefp.core.utilities.UnitaryOptimizer\_4\_2, [375](#)
- gefp.density.ci.CIS\_CIWavefunction, [127](#)
- gefp.density.ci.CIWavefunction, [135](#)
- gefp.density.ci.HF\_CIWavefunction, [251](#)
- gefp.density.ci.Reference\_SlaterDeterminant, [317](#)
- gefp.density.ci.Single\_SlaterDeterminant, [335](#)
- gefp.density.ci.SlaterDeterminant, [337](#)
- gefp.density.dfi, [87](#)
- gefp.density.dfi.DFI\_JK, [158](#)
- gefp.density.dfi.DFI\_J, [158](#)

- gefp.density.dfi.DFI, 156
- gefp.density.dfi.SCF, 329
- gefp.density.dmft.DMFT\_AO, 164
- gefp.density.dmft.DMFT\_MO, 164
- gefp.density.dmft.DMFT\_NC, 165
- gefp.density.dmft.DMFT\_PC, 165
- gefp.density.dmft.DMFT\_ProjD, 166
- gefp.density.dmft.DMFT\_ProjP, 167
- gefp.density.dmft.DMFT, 160
- gefp.density.dmft.ElectronCorrelation, 195
- gefp.density.dmft.OEProp, 291
- gefp.density.dms.\_DMS\_SCF\_Procedure, 101
- gefp.density.dms.\_Global\_Settings\_DMSFit, 101
- gefp.density.dms.Aggregate, 106
- gefp.density.dms.Basic\_DMS, 118
- gefp.density.dms.BasicD\_DMS, 119
- gefp.density.dms.Computer, 136
- gefp.density.dms.DMSFit, 169
- gefp.density.dms.DMS, 167
- gefp.density.dms.EFP\_DMSFit, 194
- gefp.density.dms.ExternalField\_EFP\_DMSFit, 209
- gefp.density.dms.Field\_DMS, 215
- gefp.density.dms.New2\_DMS, 269
- gefp.density.dms.New\_DMS, 270
- gefp.density.dms.OtherComputer, 292
- gefp.density.dms.Property\_Computer, 306
- gefp.density.dms.Rotation\_DMSFit, 327
- gefp.density.dms.SimpleComputer, 335
- gefp.density.dms.Translation\_DMSFit, 352
- gefp.density.functional.A\_V1\_MEDI\_XCFunctional, 102
- gefp.density.functional.A\_V2\_MEDI\_XCFunctional, 103
- gefp.density.functional.BBC1\_XCFunctional, 119
- gefp.density.functional.BBC2\_XCFunctional, 120
- gefp.density.functional.GU\_XCFunctional, 249
- gefp.density.functional.HF\_XCFunctional, 251
- gefp.density.functional.IDF1\_XCFunctional, 252
- gefp.density.functional.Interpolating\_XCFunctional, 254
- gefp.density.functional.JKOnly\_Interpolating\_XCFunctional, 255
- gefp.density.functional.MBB\_XCFunctional, 263
- gefp.density.functional.MEDI\_XCFunctional, 263
- gefp.density.functional.P\_V2\_MEDI\_XCFunctional, 292
- gefp.density.functional.V1\_MEDI\_XCFunctional, 379
- gefp.density.functional.V2\_MEDI\_XCFunctional, 380
- gefp.density.functional.XCFunctional, 389
- gefp.density.opdm.Density, 146
- gefp.density.opdm.DensityProjection, 154
- gefp.density.opdm.Dset\_DensityProjection, 184
- gefp.density.opdm.Pset\_DensityProjection, 307
- gefp.density.parameters.Guess, 250
- gefp.density.parameters.Matrix\_Guess, 262
- gefp.density.parameters.NC\_Guess, 268
- gefp.density.partitioning.DensityDecomposition, 150
- gefp.density.population, 88
- gefp.density.population.Loc, 262
- gefp.density.rvs.RVS, 327
- gefp.math.matrix.Superimposer, 339
- gefp.math.orthonorm.GrammSchmidt, 246
- gefp::core::driver::UniversalSurface
  - par\_descr\_fci\_sto3g\_1, 378
  - par\_descr\_fci\_sto3g\_2, 378
  - par\_full\_d\_fci\_sto3g\_1, 379
- gefp::core::utilities::UnitaryOptimizer
  - \_\_init\_\_, 368
  - maximize, 368
  - minimize, 368
  - run, 369
  - Z, 369
- gefp::core::utilities::UnitaryOptimizer\_4\_2
  - \_\_init\_\_, 376
  - maximize, 376
  - minimize, 376
  - run, 376
  - Z, 376
- gefp::density::dfi::DFI
  - run, 157
- gefp::density::dmft::DMFT
  - create, 163
  - run, 163
- gefp::density::functional::XCFunctional

- create, 390
- gefp::density::opdm::Density
  - natural\_orbitals, 148
- gefp::density::partitioning::DensityDecomposition
  - compute, 153
  - deformation\_density, 153
- gefp::density::population
  - atomic\_charges, 89
- gefp::math::matrix::Superimposer
  - \_\_init\_\_, 339
  - set, 340
- GramSchmidt
  - oepdev::GramSchmidt, 248
- include/oepdev\_files.h, 391
- include/oepdev\_options.h, 391
- index
  - oepdev::AllAOIntegralsIterator\_2, 108
  - oepdev::AllAOIntegralsIterator\_4, 110
- level
  - oepdev::MultipoleConvergence, 268
- main.cc, 392
- make\_mdh\_D1\_coeff
  - The Integral Package Library, 62
- make\_mdh\_D2\_coeff
  - The Integral Package Library, 63
- make\_mdh\_D2\_coeff\_explicit\_recursion
  - The Integral Package Library, 63
- make\_mdh\_D3\_coeff
  - The Integral Package Library, 64
- make\_mdh\_R\_coeff
  - The Integral Package Library, 65
- make\_oeps3d
  - oepdev::OEPotential, 289
- matrix
  - oepdev::GenEffPar, 229
- matrix\_power\_derivative
  - The OEPDev Utilities, 81
- maximize
  - gefp::core::utilities::UnitaryOptimizer, 368
  - gefp::core::utilities::UnitaryOptimizer\_4\_2, 376
- minimize
  - gefp::core::utilities::UnitaryOptimizer, 368
  - gefp::core::utilities::UnitaryOptimizer\_4\_2, 376
- MultipoleConvergence
  - oepdev::DMTPole, 181
  - oepdev::MultipoleConvergence, 267
- natural\_orbitals
  - gefp::density::opdm::Density, 148
- nmo\_
  - oepdev::CISComputer, 134
- OEPDevSolver
  - oepdev::OEPDevSolver, 283
- OEPotential
  - oepdev::OEPotential, 287, 288
- OEPotential3D
  - The Three-Dimensional Vector Fields Library, 68
- oepdev, 89
  - psi, 98
- oepdev/lib3d/dmtp.h, 393
- oepdev/lib3d/esp.h, 393
- oepdev/libgefp/gefp.h, 394
- oepdev/libints/eri.h, 395
- oepdev/libints/recurr.h, 396
- oepdev/liboep/oep.h, 397
- oepdev/liboep/oep\_gdf.h, 398
- oepdev/libpsi/integral.h, 399
- oepdev/libpsi/potential.h, 399
- oepdev/libsolver/solver.h, 400
- oepdev/libsolver/ti\_data.h, 401
- oepdev/libtest/test.h, 401
- oepdev/libutil/basis\_rotation.h, 402
- oepdev/libutil/cis.h, 405
- oepdev/libutil/davidson\_liu.h, 406
- oepdev/libutil/diis.h, 406
- oepdev/libutil/gram\_schmidt.h, 407
- oepdev/libutil/integrals\_iter.h, 407
- oepdev/libutil/kabsch\_superimposer.h, 408
- oepdev/libutil/scf\_perturb.h, 408
- oepdev/libutil/unitary\_optimizer.h, 409
- oepdev/libutil/util.h, 410
- oepdev/libutil/wavefunction\_union.h, 412
- oepdev::ABCD, 104
- oepdev::AOIntegralsIterator, 116
  - build, 117
- oepdev::AbInitioPolarGEFactory, 105
- oepdev::AllAOIntegralsIterator\_2, 106
  - AllAOIntegralsIterator\_2, 107
  - index, 108
- oepdev::AllAOIntegralsIterator\_4, 108
  - AllAOIntegralsIterator\_4, 109

- index, [110](#)
- oepdev::AllAOShellCombinationsIterator\_2, [110](#)
  - AllAOShellCombinationsIterator\_2, [111](#), [112](#)
  - compute\_shell, [112](#)
- oepdev::AllAOShellCombinationsIterator\_4, [113](#)
  - AllAOShellCombinationsIterator\_4, [114](#), [115](#)
  - compute\_shell, [115](#)
- oepdev::CAMM, [121](#)
- oepdev::CISComputer, [127](#)
  - build, [130](#)
  - nmo\_, [134](#)
- oepdev::CISData, [134](#)
- oepdev::CPHF, [137](#)
  - CPHF, [140](#)
- oepdev::ChargeTransferEnergyOEPotential, [122](#)
- oepdev::ChargeTransferEnergySolver, [123](#)
  - compute\_benchmark, [126](#)
  - compute\_oep\_based, [126](#)
- oepdev::CubePoints3DIterator, [140](#)
- oepdev::CubePointsCollection3D, [142](#)
- oepdev::DIISManager, [159](#)
  - compute, [160](#)
  - DIISManager, [159](#)
  - put, [160](#)
  - update, [160](#)
- oepdev::DMTPole, [170](#)
  - build, [176](#)
  - compute, [176](#)
  - DMTPole, [175](#)
  - determine\_dmtip\_convergence\_level, [176](#)
  - empty, [177](#)
  - energy, [177](#)
  - field, [178](#)
  - MultipoleConvergence, [181](#)
  - potential, [178](#)
  - recenter, [179](#)
  - rotate, [180](#)
  - superimpose, [181](#)
- oepdev::DavidsonLiu, [142](#)
- oepdev::DoubleGeneralizedDensityFit, [182](#)
  - compute, [184](#)
- oepdev::EETCouplingOEPotential, [185](#)
- oepdev::EETCouplingSolver, [186](#)
  - compute\_benchmark, [192](#)
  - compute\_oep\_based, [192](#)
- oepdev::EFP2\_GEFactory, [193](#)
- oepdev::ERI\_1\_1, [202](#)
- oepdev::ERI\_2\_2, [203](#)
- oepdev::ERI\_3\_1, [205](#)
- oepdev::ESPSolver, [206](#)
  - ESPSolver, [208](#), [209](#)
- oepdev::ElectrostaticEnergyOEPotential, [195](#)
- oepdev::ElectrostaticEnergySolver, [196](#)
  - compute\_benchmark, [199](#)
  - compute\_oep\_based, [199](#)
- oepdev::ElectrostaticPotential3D, [200](#)
- oepdev::FFAbInitioPolarGEFactory, [210](#)
- oepdev::Field3D, [211](#)
  - build, [214](#)
  - Field3D, [213](#)
- oepdev::Fourier9, [216](#)
- oepdev::GenEffFrag, [216](#)
  - compute\_energy, [220](#)
  - compute\_many\_body\_energy, [220](#)
  - energy, [220](#)
  - susceptibility, [221](#)
- oepdev::GenEffPar, [222](#)
  - allocate, [226](#)
  - basisset, [227](#)
  - compute\_density\_matrix, [227](#), [228](#)
  - dmtip, [228](#)
  - dpol, [229](#)
  - matrix, [229](#)
  - rotate, [229](#)
  - set\_basisset, [229](#)
  - set\_dmtip, [230](#)
  - set\_dpol, [230](#)
  - set\_matrix, [230](#)
  - set\_susceptibility, [231](#)
  - set\_vector, [231](#)
  - superimpose, [231](#)
  - susceptibility, [232](#), [233](#)
  - translate, [233](#)
  - vector, [234](#)
- oepdev::GenEffParFactory, [234](#)
  - build, [236](#)
- oepdev::GeneralizedDensityFit, [238](#)
  - build, [239](#), [240](#)
  - compute, [240](#)
- oepdev::GeneralizedPolarGEFactory, [241](#)
- oepdev::GeneralizedPolarGEFactory::StatisticalSet,



- 338
- oepdev::GramSchmidt, 247
  - GramSchmidt, 248
  - projection, 249
- oepdev::IntegralFactory, 253
- oepdev::KabschSuperimposer, 256
  - compute, 258
- oepdev::LinearGradientNonUniformEFieldPolarGEFactory, 258
- oepdev::LinearNonUniformEFieldPolarGEFactory, 260
- oepdev::LinearUniformEFieldPolarGEFactory, 261
- oepdev::MultipoleConvergence, 264
  - compute, 267
  - ConvergenceLevel, 266
  - level, 268
  - MultipoleConvergence, 267
  - Property, 266
- oepdev::NonUniformEFieldPolarGEFactory, 270
- oepdev::OEP\_EFP2\_GEFactory, 273
- oepdev::OEPDevSolver, 275
  - build, 283
  - compute\_benchmark, 283
  - compute\_oep\_based, 284
  - OEPDevSolver, 283
- oepdev::OEPTYPE, 291
- oepdev::OEPotential, 284
  - build, 288
  - make\_oeps3d, 289
  - OEPotential, 287, 288
- oepdev::OEPotential3D< T >, 289
- oepdev::PerturbCharges, 294
- oepdev::Points3DIterator, 295
  - build, 297, 298
  - Points3DIterator, 297
- oepdev::Points3DIterator::Point, 295
- oepdev::PointsCollection3D, 298
  - build, 300, 301
  - PointsCollection3D, 300
- oepdev::PolarGEFactory, 302
- oepdev::PotentialInt, 303
  - PotentialInt, 304, 305
  - set\_charge\_field, 305
- oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory, 307
- oepdev::QuadraticNonUniformEFieldPolarGEFactory, 309
- oepdev::QuadraticUniformEFieldPolarGEFactory, 310
- oepdev::R\_CISComputer, 311
- oepdev::R\_CISComputer\_Direct, 311
- oepdev::R\_CISComputer\_DL, 312
- oepdev::R\_CISComputer\_Explicit, 314
- oepdev::RHF Perturbed, 325
- oepdev::RandomPoints3DIterator, 315
- oepdev::RandomPointsCollection3D, 316
- oepdev::RepulsionEnergyOEPotential, 317
- oepdev::RepulsionEnergySolver, 318
  - compute\_benchmark, 324
  - compute\_oep\_based, 324
- oepdev::ShellCombinationsIterator, 330
  - ao\_iterator, 332
  - build, 333
  - compute\_shell, 334
  - ShellCombinationsIterator, 332
- oepdev::SingleGeneralizedDensityFit, 336
  - compute, 337
- oepdev::TIDData, 343
  - coupling\_direct, 347
  - coupling\_direct\_coul, 347
  - coupling\_direct\_exch, 347
  - coupling\_indirect, 348
  - coupling\_indirect\_ti2, 348
  - coupling\_indirect\_ti3, 348
  - coupling\_total, 348
  - coupling\_trcamm, 349
  - overlap\_corrected, 349
  - overlap\_corrected\_direct, 350
  - overlap\_corrected\_indirect, 351
  - v0, 351
- oepdev::TwoBodyAOInt, 353
  - compute, 354
- oepdev::TwoElectronInt, 354
  - compute\_shell, 356
- oepdev::U\_CISComputer, 357
- oepdev::U\_CISComputer\_DL, 358
- oepdev::U\_CISComputer\_Explicit, 359
- oepdev::UniformEFieldPolarGEFactory, 360
- oepdev::UnitaryOptimizer, 361
  - UnitaryOptimizer, 366
- oepdev::UnitaryOptimizer\_4\_2, 369
- oepdev::UnitaryOptimizer\_4\_2, 374
- oepdev::UnitaryTransformedMOPolarGEFactory, 377

- oepdev::WavefunctionUnion, [382](#)
  - Ca\_subset, [388](#)
  - Cb\_subset, [388](#)
  - WavefunctionUnion, [387](#)
- oepdev::test::Test, [340](#)
- overlap\_corrected
  - oepdev::TIData, [349](#)
- overlap\_corrected\_direct
  - oepdev::TIData, [350](#)
- overlap\_corrected\_indirect
  - oepdev::TIData, [351](#)
- par\_descr\_fci\_sto3g\_1
  - gefp::core::driver::UniversalSurface, [378](#)
- par\_descr\_fci\_sto3g\_2
  - gefp::core::driver::UniversalSurface, [378](#)
- par\_full\_d\_fci\_sto3g\_1
  - gefp::core::driver::UniversalSurface, [379](#)
- Points3DIterator
  - oepdev::Points3DIterator, [297](#)
- PointsCollection3D
  - oepdev::PointsCollection3D, [300](#)
- populate
  - The OEPDev Utilities, [82](#)
- potential
  - oepdev::DMTPole, [178](#)
- PotentialInt
  - oepdev::PotentialInt, [304](#), [305](#)
- projection
  - oepdev::GramSchmidt, [249](#)
- Property
  - oepdev::MultipoleConvergence, [266](#)
- psi, [98](#)
  - oepdev, [98](#)
  - read\_options, [99](#)
- put
  - oepdev::DIISManager, [160](#)
- r6
  - The OEPDev Utilities, [83](#)
- read\_options
  - psi, [99](#)
- recenter
  - oepdev::DMTPole, [179](#)
- rotate
  - oepdev::DMTPole, [180](#)
  - oepdev::GenEffPar, [229](#)
- run
  - gefp::core::utilities::UnitaryOptimizer, [369](#)
  - gefp::core::utilities::UnitaryOptimizer\_4\_2, [376](#)
  - gefp::density::dfi::DFI, [157](#)
  - gefp::density::dmft::DMFT, [163](#)
- set
  - gefp::math::matrix::Superimposer, [340](#)
- set\_basisset
  - oepdev::GenEffPar, [229](#)
- set\_charge\_field
  - oepdev::PotentialInt, [305](#)
- set\_dmtip
  - oepdev::GenEffPar, [230](#)
- set\_dpole
  - oepdev::GenEffPar, [230](#)
- set\_matrix
  - oepdev::GenEffPar, [230](#)
- set\_susceptibility
  - oepdev::GenEffPar, [231](#)
- set\_vector
  - oepdev::GenEffPar, [231](#)
- ShellCombinationsIterator
  - oepdev::ShellCombinationsIterator, [332](#)
- solve\_scf
  - The OEPDev Utilities, [83](#)
- solve\_scf\_sad
  - The OEPDev Utilities, [84](#)
- superimpose
  - oepdev::DMTPole, [181](#)
  - oepdev::GenEffPar, [231](#)
- susceptibility
  - oepdev::GenEffFrag, [221](#)
  - oepdev::GenEffPar, [232](#), [233](#)
- The Density Functional Theory Library, [70](#)
- The Generalized Effective Fragment Potentials Library, [55](#)
- The Generalized One-Electron Potentials Library, [53](#)
- The Integral Package Library, [57](#)
  - d\_N\_n1\_n2, [62](#)
  - make\_mdh\_D1\_coeff, [62](#)
  - make\_mdh\_D2\_coeff, [63](#)
  - make\_mdh\_D2\_coeff\_explicit\_recursion, [63](#)
  - make\_mdh\_D3\_coeff, [64](#)
  - make\_mdh\_R\_coeff, [65](#)
- The OEPDev Solver Library, [54](#)
- The OEPDev Testing Platform Library, [86](#)
- The OEPDev Utilities, [71](#)



- [\\_calculate\\_DFI\\_Vel](#), [76](#)
  - [ao\\_rotation\\_matrix](#), [77](#)
  - [average\\_moment](#), [77](#)
  - [calculate\\_DFI\\_Vel\\_JK](#), [78](#)
  - [calculate\\_DFI\\_Vel\\_J](#), [78](#)
  - [calculate\\_JK\\_r](#), [80](#)
  - [calculate\\_JK](#), [79](#)
  - [calculate\\_der\\_D](#), [77](#)
  - [calculate\\_e\\_xc](#), [79](#)
  - [compute\\_distance](#), [80](#)
  - [create\\_superfunctional](#), [81](#)
  - [extract\\_monomer](#), [81](#)
  - [matrix\\_power\\_derivative](#), [81](#)
  - [populate](#), [82](#)
  - [r6](#), [83](#)
  - [solve\\_scf](#), [83](#)
  - [solve\\_scf\\_sad](#), [84](#)
- The Three-Dimensional Vector Fields Library,  
[67](#)
  - [OEPotential3D](#), [68](#)
- translate
  - [oepdev::GenEffPar](#), [233](#)
- UnitaryOptimizer
  - [oepdev::UnitaryOptimizer](#), [366](#)
- UnitaryOptimizer\_4\_2
  - [oepdev::UnitaryOptimizer\\_4\\_2](#), [374](#)
- update
  - [oepdev::DIISManager](#), [160](#)
- v0
  - [oepdev::TIDData](#), [351](#)
- vector
  - [oepdev::GenEffPar](#), [234](#)
- WavefunctionUnion
  - [oepdev::WavefunctionUnion](#), [387](#)
- Z
  - [gefp::core::utilities::UnitaryOptimizer](#), [369](#)
  - [gefp::core::utilities::UnitaryOptimizer\\_4\\_2](#),  
[376](#)