

OPPLA — OPTimality-based PLAnkton ecosystem model

Generated by Doxygen 1.9.7

1 Modules Index	1
1.1 Modules List	1
2 Data Type Index	3
2.1 Class Hierarchy	3
3 Data Type Index	5
3.1 Data Types List	5
4 File Index	7
4.1 File List	7
5 Module Documentation	9
5.1 bac Module Reference	9
5.1.1 Detailed Description	9
5.1.2 Function/Subroutine Documentation	10
5.1.2.1 bac_flux()	10
5.1.2.2 bac_grow()	10
5.1.2.3 bac_read()	10
5.1.2.4 bac_set()	11
5.1.2.5 bac_uptake()	11
5.1.2.6 bac_uptake_ft()	12
5.1.2.7 dom_flux()	12
5.1.2.8 dom_read()	12
5.1.2.9 dom_set()	12
5.2 brock81 Module Reference	13
5.2.1 Detailed Description	14
5.2.2 Function/Subroutine Documentation	14
5.2.2.1 brock81_init()	14
5.2.2.2 csrad()	14
5.2.2.3 daylength()	15
5.2.2.4 declination_b81()	16
5.2.2.5 declination_f95()	16
5.2.2.6 par_brock81()	17
5.2.2.7 par_ld()	18
5.2.2.8 par_mesoaqu()	18
5.2.2.9 rdrad()	19
5.2.2.10 sunday_brock81()	19
5.2.2.11 sunday_const()	20
5.2.2.12 sunday_ld()	21
5.2.2.13 sunday_mesoaqu()	21
5.2.3 Variable Documentation	22
5.2.3.1 cfhds	22
5.2.3.2 d15	22

5.2.3.3 fd	22
5.2.3.4 sdl	22
5.2.3.5 solar	22
5.2.3.6 tilt	23
5.2.3.7 yeardays	23
5.3 cfo Module Reference	23
5.3.1 Detailed Description	24
5.3.2 Data Type Documentation	24
5.3.2.1 type cfo::pcc	24
5.3.3 Function/Subroutine Documentation	24
5.3.3.1 cfo_dvm()	24
5.3.3.2 cfo_flux()	25
5.3.3.3 cfo_read()	25
5.3.3.4 cfo_set()	26
5.3.3.5 cfo_svm_dyn()	27
5.3.3.6 cfo_svm_static()	27
5.3.3.7 egest_dm()	28
5.3.3.8 egest_pm()	29
5.3.3.9 excrete_dim()	29
5.3.3.10 excrete_dom()	30
5.3.3.11 foract()	30
5.3.3.12 forage_cfo()	30
5.3.3.13 forage_switch()	31
5.4 clops Module Reference	32
5.4.1 Detailed Description	32
5.4.2 Function/Subroutine Documentation	32
5.4.2.1 getopt()	32
5.5 cmo Module Reference	33
5.5.1 Detailed Description	34
5.5.2 Function/Subroutine Documentation	34
5.5.2.1 chl_dyn()	34
5.5.2.2 cmo_flux()	34
5.5.2.3 cmo_pic()	35
5.5.2.4 cmo_read()	35
5.5.2.5 cmo_set()	36
5.5.2.6 facn2f()	37
5.5.2.7 ftnf_eppley()	37
5.5.2.8 ftnf_houlton()	38
5.5.2.9 ftnf_oppla()	38
5.5.2.10 grow()	38
5.5.2.11 tchdyn()	40
5.5.2.12 tchia()	40

5.5.2.13 uptake()	41
5.6 det Module Reference	41
5.6.1 Detailed Description	41
5.6.2 Function/Subroutine Documentation	42
5.6.2.1 det_flux()	42
5.6.2.2 det_fluxes()	42
5.6.2.3 det_loss_pic()	42
5.6.2.4 det_read()	43
5.6.2.5 det_set()	43
5.7 dic Module Reference	43
5.7.1 Detailed Description	45
5.7.2 Function/Subroutine Documentation	46
5.7.2.1 asfco2()	46
5.7.2.2 co2()	47
5.7.2.3 dic_potalk()	49
5.7.2.4 dicalp()	50
5.7.2.5 eos80_rho()	52
5.7.2.6 faco2()	52
5.7.2.7 hini_f06()	53
5.7.2.8 hini_m13()	54
5.7.2.9 hsolve_f06()	55
5.7.2.10 hsolve_m13()	56
5.7.2.11 indic()	57
5.7.2.12 k0co2()	58
5.7.2.13 lgkspa()	59
5.7.2.14 lgkspc()	60
5.7.2.15 lnk1_sws()	60
5.7.2.16 lnk1_total()	61
5.7.2.17 lnk1p_sws()	61
5.7.2.18 lnk2_sws()	62
5.7.2.19 lnk2_total()	62
5.7.2.20 lnk2p_sws()	63
5.7.2.21 lnk3p_sws()	63
5.7.2.22 lnkb_total()	64
5.7.2.23 lnkf_free()	65
5.7.2.24 lnkf_total()	65
5.7.2.25 lnknh4_sws()	66
5.7.2.26 lnks_free()	66
5.7.2.27 lnkw_sws()	67
5.7.2.28 pivel_lm86()	68
5.7.2.29 pivel_w14()	68
5.7.2.30 pivel_w92()	69

5.7.2.31 pivel_w99()	70
5.7.3 Variable Documentation	70
5.7.3.1 rgas	70
5.7.3.2 t0ck	71
5.8 dnode Module Reference	71
5.8.1 Detailed Description	71
5.8.2 Function/Subroutine Documentation	71
5.8.2.1 read_ode()	71
5.8.2.2 set_atol()	72
5.9 et Module Reference	72
5.9.1 Detailed Description	73
5.9.2 Data Type Documentation	73
5.9.2.1 type et::layer	73
5.9.2.2 type et::local	74
5.9.2.3 type et::state	77
5.9.2.4 type et::timing	77
5.9.3 Function/Subroutine Documentation	78
5.9.3.1 ft_eppley()	78
5.9.3.2 ft_houlton()	78
5.9.3.3 ft_me()	79
5.9.3.4 ft_oppla()	79
5.9.3.5 ft_select()	79
5.9.4 Variable Documentation	80
5.9.4.1 constituent_units	80
5.9.4.2 constituents	80
5.9.4.3 pi	81
5.9.4.4 rad	81
5.10 fudu Module Reference	81
5.10.1 Detailed Description	82
5.10.2 Data Type Documentation	82
5.10.2.1 type fudu::quantity	82
5.10.3 Function/Subroutine Documentation	83
5.10.3.1 convert()	83
5.10.3.2 exudu()	83
5.10.3.3 inudu()	84
5.10.3.4 slot()	84
5.10.3.5 uduerr()	85
5.10.4 Variable Documentation	85
5.10.4.1 ascii	85
5.10.4.2 cfds	85
5.10.4.3 cfhs	86
5.10.4.4 latin1	86

5.10.4.5	ute_bad_arg	86
5.10.4.6	ute_cant_format	86
5.10.4.7	ute_exists	86
5.10.4.8	ute_meaningless	86
5.10.4.9	ute_no_second	86
5.10.4.10	ute_no_unit	86
5.10.4.11	ute_not_same_system	87
5.10.4.12	ute_open_arg	87
5.10.4.13	ute_open_default	87
5.10.4.14	ute_open_env	87
5.10.4.15	ute_os	87
5.10.4.16	ute_parse	87
5.10.4.17	ute_success	87
5.10.4.18	ute_syntax	87
5.10.4.19	ute_unknown	88
5.10.4.20	ute_visit_error	88
5.10.4.21	utencoding	88
5.10.4.22	utf8	88
5.10.4.23	utsystem	88
5.11	julian Module Reference	88
5.11.1	Detailed Description	90
5.11.2	Data Type Documentation	90
5.11.2.1	type julian::leap	90
5.11.2.2	type julian::tm_struct	90
5.11.2.3	type julian::yms	91
5.11.3	Function/Subroutine Documentation	91
5.11.3.1	jul_dhmsfsec()	91
5.11.3.2	jul_dsofsec()	92
5.11.3.3	jul_dutcofynd()	93
5.11.3.4	jul_fixym()	93
5.11.3.5	jul_formatdate()	94
5.11.3.6	jul_gregymdofjd()	94
5.11.3.7	jul_initleaps()	95
5.11.3.8	jul_isleapday()	95
5.11.3.9	jul_jdofgregymd()	96
5.11.3.10	jul_jdofjulymd()	96
5.11.3.11	jul_julymdofjd()	96
5.11.3.12	jul_leapsecs()	97
5.11.3.13	jul_leapsecsym()	98
5.11.3.14	jul_parsedt()	98
5.11.3.15	jul_taiofdutc()	99
5.11.3.16	jul_taiofet()	100

5.11.3.17 jul_taiofjd()	101
5.11.3.18 jul_ydofdutc()	102
5.11.3.19 jul_ymdofdutc()	103
5.11.3.20 leap_index()	104
5.11.4 Variable Documentation	104
5.11.4.1 dp	104
5.11.4.2 gregorian_day	104
5.11.4.3 gregorian_dutc	104
5.11.4.4 gregorian_month	104
5.11.4.5 gregorian_year	105
5.11.4.6 jd_of_j2000_noon	105
5.11.4.7 jdn_of_j2000	105
5.11.4.8 jul_et_type	105
5.11.4.9 jul_tai_type	105
5.11.4.10 jul_utc_type	105
5.11.4.11 leap_defaults	106
5.11.4.12 leap_table	106
5.11.4.13 leaps	106
5.11.4.14 mjd_of_j2000_noon	106
5.12 lambert Module Reference	106
5.12.1 Detailed Description	107
5.12.2 Function/Subroutine Documentation	107
5.12.2.1 lwm1()	107
5.12.2.2 wapd()	107
5.12.2.3 wapr()	108
5.12.3 Variable Documentation	108
5.12.3.1 dp	108
5.13 onf Module Reference	108
5.13.1 Detailed Description	109
5.13.2 Data Type Documentation	109
5.13.2.1 type onf::dataset	109
5.13.2.2 type onf::nf90stratt	110
5.13.3 Function/Subroutine Documentation	110
5.13.3.1 exof()	110
5.13.3.2 innf()	110
5.13.3.3 inpd()	111
5.13.3.4 invar()	111
5.13.3.5 nf90info()	113
5.13.3.6 onf_error()	113
5.13.3.7 outsv()	114
5.13.3.8 read_dim()	114
5.13.3.9 read_ds()	115

5.13.3.10 readpd()	116
5.13.3.11 write_output()	116
5.13.4 Variable Documentation	117
5.13.4.1 bfan	117
5.13.4.2 dfan	117
5.13.4.3 ico2ds	117
5.13.4.4 infan	117
5.13.4.5 iprsds	117
5.13.4.6 latan	117
5.13.4.7 lonan	118
5.13.4.8 nbvds	118
5.13.4.9 ngdds	118
5.13.4.10 npfds	118
5.13.4.11 nsfds	118
5.13.4.12 nsvds	118
5.13.4.13 offsets	119
5.13.4.14 pdgds	119
5.13.4.15 pdsds	119
5.13.4.16 pfan	119
5.13.4.17 plfan	119
5.13.4.18 sufan	119
5.13.4.19 timan	120
5.13.4.20 timunt	120
5.14 plankton Module Reference	120
5.14.1 Detailed Description	121
5.14.2 Data Type Documentation	121
5.14.2.1 type plankton::bc	121
5.14.3 Function/Subroutine Documentation	121
5.14.3.1 aggregate()	121
5.14.3.2 alkalinity()	122
5.14.3.3 boxbc_cd()	123
5.14.3.4 boxbc_upwind()	123
5.14.3.5 fpar_vertical()	124
5.14.3.6 lch_chl()	124
5.14.3.7 lch_pon()	125
5.14.3.8 plankton_init()	125
5.14.3.9 plankton_ode()	127
5.14.4 Variable Documentation	128
5.14.4.1 bacpla	128
5.14.4.2 box	128
5.14.4.3 detrit	128
5.14.4.4 dics	128

5.14.4.5 doma	128
5.14.4.6 envir	129
5.14.4.7 fungrp	129
5.14.4.8 parode	129
5.14.4.9 phypla	129
5.14.4.10 phys	129
5.14.4.11 states	129
5.14.4.12 times	130
5.14.4.13 zoopla	130
5.15 stdunt Module Reference	130
5.15.1 Detailed Description	130
5.15.2 Variable Documentation	130
5.15.2.1 dp	130
5.15.2.2 stderr	130
5.15.2.3 stdin	130
5.15.2.4 stdout	130
6 Data Type Documentation	131
6.1 et::aggregation Interface Reference	131
6.1.1 Detailed Description	131
6.1.2 Constructor & Destructor Documentation	131
6.1.2.1 aggregation()	131
6.2 et::attenuate Interface Reference	131
6.2.1 Detailed Description	132
6.2.2 Constructor & Destructor Documentation	132
6.2.2.1 attenuate()	132
6.3 bac::bacteria Type Reference	132
6.3.1 Detailed Description	135
6.3.2 Member Function/Subroutine Documentation	135
6.3.2.1 flux()	135
6.3.2.2 read()	135
6.3.2.3 set()	135
6.3.3 Member Data Documentation	136
6.3.3.1 adim	136
6.3.3.2 adop	136
6.3.3.3 doc	136
6.3.3.4 docnp	136
6.3.3.5 dom	136
6.3.3.6 don	136
6.3.3.7 dop	137
6.3.3.8 ftem	137
6.3.3.9 ggem	137

6.3.3.10 grow	137
6.3.3.11 idocnp	137
6.3.3.12 idom	137
6.3.3.13 lambda	137
6.3.3.14 n2p	138
6.3.3.15 qdon	138
6.3.3.16 qdop	138
6.3.3.17 qn_param	138
6.3.3.18 qp_param	138
6.3.3.19 rc	138
6.3.3.20 uptake	138
6.3.3.21 vdin	139
6.3.3.22 vdip	139
6.3.3.23 vdoc	139
6.3.3.24 vdom	139
6.3.3.25 vdon	139
6.3.3.26 vdop	139
6.3.3.27 vm	139
6.3.3.28 vmax	139
6.3.3.29 zeta	140
6.4 et::boxalk Interface Reference	140
6.4.1 Detailed Description	140
6.4.2 Constructor & Destructor Documentation	140
6.4.2.1 boxalk()	140
6.5 et::boxbc Interface Reference	140
6.5.1 Detailed Description	140
6.5.2 Constructor & Destructor Documentation	141
6.5.2.1 boxbc()	141
6.6 julian::c_strftime Interface Reference	141
6.6.1 Detailed Description	141
6.6.2 Constructor & Destructor Documentation	141
6.6.2.1 c_strftime()	141
6.7 julian::c_strptime Interface Reference	142
6.7.1 Detailed Description	142
6.7.2 Constructor & Destructor Documentation	142
6.7.2.1 c_strptime()	142
6.8 fudu::cv_convert Interface Reference	142
6.8.1 Detailed Description	142
6.8.2 Member Function/Subroutine Documentation	143
6.8.2.1 cv_convert_double()	143
6.8.2.2 cv_convert_float()	143
6.9 fudu::cv_free Interface Reference	143

6.9.1 Detailed Description	143
6.9.2 Constructor & Destructor Documentation	143
6.9.2.1 cv_free()	143
6.10 et::decl Interface Reference	144
6.10.1 Detailed Description	144
6.10.2 Constructor & Destructor Documentation	144
6.10.2.1 decl()	144
6.11 det::detritus Type Reference	144
6.11.1 Detailed Description	147
6.11.2 Member Function/Subroutine Documentation	147
6.11.2.1 flux()	147
6.11.2.2 read()	147
6.11.2.3 set()	148
6.11.3 Member Data Documentation	148
6.11.3.1 aggregates	148
6.11.3.2 decay	148
6.11.3.3 detchl	148
6.11.3.4 detpic	148
6.11.3.5 fluxes	149
6.11.3.6 formpic	149
6.11.3.7 iloss	149
6.11.3.8 irec	149
6.11.3.9 isvl	149
6.11.3.10 kag	149
6.11.3.11 loss	150
6.11.3.12 losses	150
6.11.3.13 nrec	150
6.11.3.14 omax	150
6.11.3.15 remi	150
6.11.3.16 remic	150
6.11.3.17 remichl	150
6.11.3.18 remin	151
6.11.3.19 remip	151
6.11.3.20 remipic	151
6.11.3.21 sink	151
6.12 dic::dicsys Type Reference	151
6.12.1 Detailed Description	153
6.12.2 Member Function/Subroutine Documentation	153
6.12.2.1 asf()	153
6.12.2.2 co2()	153
6.12.2.3 read()	154
6.12.3 Member Data Documentation	155

6.12.3.1 ac	155
6.12.3.2 acd	155
6.12.3.3 alkalinity	155
6.12.3.4 asfco2	155
6.12.3.5 asfo2	155
6.12.3.6 at	155
6.12.3.7 bt	155
6.12.3.8 ca	156
6.12.3.9 ct	156
6.12.3.10 faco2	156
6.12.3.11 ft	156
6.12.3.12 gmk	156
6.12.3.13 hsolve	156
6.12.3.14 init	156
6.12.3.15 k0co2	156
6.12.3.16 k1	157
6.12.3.17 k12	157
6.12.3.18 k12p	157
6.12.3.19 k1p	157
6.12.3.20 k2	157
6.12.3.21 k2p	157
6.12.3.22 k3p	157
6.12.3.23 kb	157
6.12.3.24 kf	158
6.12.3.25 knh4	158
6.12.3.26 ksi	158
6.12.3.27 kso4	158
6.12.3.28 kspa	158
6.12.3.29 kspc	158
6.12.3.30 kw	158
6.12.3.31 niter	158
6.12.3.32 ph0	159
6.12.3.33 pivel	159
6.12.3.34 potalk	159
6.12.3.35 pt	159
6.12.3.36 r_tot_free	159
6.12.3.37 r_tot_sws	159
6.12.3.38 rrnp	160
6.12.3.39 rrnsi	160
6.12.3.40 schnco2	160
6.12.3.41 schno2	160
6.12.3.42 sit	160

6.12.3.43 so4	160
6.12.3.44 tk	160
6.12.3.45 totalk	161
6.12.3.46 tvco2	161
6.12.3.47 tvo2	161
6.13 onf::dimension Type Reference	161
6.13.1 Detailed Description	161
6.13.2 Member Function/Subroutine Documentation	161
6.13.2.1 read()	161
6.13.3 Member Data Documentation	162
6.13.3.1 bounds	162
6.13.3.2 id	162
6.13.3.3 name	162
6.13.3.4 units	162
6.13.3.5 values	162
6.13.3.6 varid	163
6.14 bac::dom Type Reference	163
6.14.1 Detailed Description	165
6.14.2 Member Function/Subroutine Documentation	165
6.14.2.1 flux()	165
6.14.2.2 read()	165
6.14.2.3 set()	166
6.14.3 Member Data Documentation	166
6.14.3.1 ldc	166
6.14.3.2 ldln	166
6.14.3.3 relac	166
6.14.3.4 relan	166
6.14.3.5 trlc	166
6.14.3.6 trln	167
6.15 et::fluxes Interface Reference	167
6.15.1 Detailed Description	167
6.15.2 Constructor & Destructor Documentation	167
6.15.2.1 fluxes()	167
6.16 et::fpar Interface Reference	167
6.16.1 Detailed Description	168
6.16.2 Constructor & Destructor Documentation	168
6.16.2.1 fpar()	168
6.17 et::fungroup Type Reference	168
6.17.1 Detailed Description	169
6.17.2 Member Function/Subroutine Documentation	170
6.17.2.1 flux()	170
6.17.2.2 ft_select()	170

6.17.2.3 read()	170
6.17.2.4 set()	171
6.17.3 Member Data Documentation	171
6.17.3.1 constituents	171
6.17.3.2 ft	171
6.17.3.3 icon	171
6.17.3.4 ifg	171
6.17.3.5 iq0	172
6.17.3.6 ir	172
6.17.3.7 isv	172
6.17.3.8 motile	172
6.17.3.9 name	172
6.17.3.10 names	172
6.17.3.11 ncon	173
6.17.3.12 nq0	173
6.17.3.13 nsv	173
6.17.3.14 oc	173
6.17.3.15 q0	173
6.17.3.16 q10	173
6.17.3.17 qn	174
6.17.3.18 qp	174
6.17.3.19 ratios	174
6.17.3.20 stick	174
6.17.3.21 sticky	174
6.17.3.22 topt	174
6.17.3.23 tref	174
6.17.3.24 tspr	175
6.17.3.25 units	175
6.17.3.26 vv	175
6.18 et::init Interface Reference	175
6.18.1 Detailed Description	175
6.18.2 Constructor & Destructor Documentation	175
6.18.2.1 init()	175
6.19 lambert::lambertw Interface Reference	176
6.19.1 Detailed Description	176
6.19.2 Member Function/Subroutine Documentation	176
6.19.2.1 wapd()	176
6.19.2.2 wapr()	176
6.20 et::light Interface Reference	177
6.20.1 Detailed Description	177
6.20.2 Constructor & Destructor Documentation	177
6.20.2.1 light()	177

6.21 onf::nf90grd Type Reference	177
6.21.1 Detailed Description	178
6.21.2 Member Data Documentation	178
6.21.2.1 values	178
6.22 onf::nf90group Type Reference	179
6.22.1 Detailed Description	179
6.22.2 Member Function/Subroutine Documentation	180
6.22.2.1 info()	180
6.22.3 Member Data Documentation	180
6.22.3.1 attributes	180
6.22.3.2 bounds	180
6.22.3.3 depth	181
6.22.3.4 depth_flux	181
6.22.3.5 filename	181
6.22.3.6 id	181
6.22.3.7 ilat	181
6.22.3.8 ilon	181
6.22.3.9 it0	181
6.22.3.10 lat	181
6.22.3.11 lon	182
6.22.3.12 nbox	182
6.22.3.13 ntim	182
6.22.3.14 state	182
6.22.3.15 time	182
6.22.3.16 timediff	182
6.23 onf::nf90var Type Reference	183
6.23.1 Detailed Description	183
6.23.2 Member Function/Subroutine Documentation	183
6.23.2.1 read()	183
6.23.3 Member Data Documentation	184
6.23.3.1 name	184
6.23.3.2 ndims	184
6.23.3.3 type	184
6.23.3.4 units	184
6.23.3.5 varid	184
6.24 onf::nf90vec Type Reference	185
6.24.1 Detailed Description	186
6.24.2 Member Data Documentation	186
6.24.2.1 values	186
6.25 dcode::ode Type Reference	186
6.25.1 Detailed Description	187
6.25.2 Member Function/Subroutine Documentation	187

6.25.2.1 read()	187
6.25.2.2 set_atol()	187
6.25.3 Member Data Documentation	188
6.25.3.1 atol	188
6.25.3.2 clower	188
6.25.3.3 constr	188
6.25.3.4 constrained	188
6.25.3.5 cupper	188
6.25.3.6 h0	188
6.25.3.7 hmax	189
6.25.3.8 hmin	189
6.25.3.9 itask	189
6.25.3.10 jsv	189
6.25.3.11 maxord	189
6.25.3.12 meth	189
6.25.3.13 mf	189
6.25.3.14 miter	189
6.25.3.15 moss	190
6.25.3.16 mxatol	190
6.25.3.17 mxhnil	190
6.25.3.18 mxrtol	190
6.25.3.19 mxstep	190
6.25.3.20 names	190
6.25.3.21 nconstr	190
6.25.3.22 neq	191
6.25.3.23 quiet	191
6.25.3.24 rtol	191
6.25.3.25 sparse_jacobian	191
6.26 cmo::phycmo Type Reference	191
6.26.1 Detailed Description	196
6.26.2 Member Function/Subroutine Documentation	196
6.26.2.1 flux()	196
6.26.2.2 read()	196
6.26.2.3 set()	196
6.26.3 Member Data Documentation	197
6.26.3.1 a	197
6.26.3.2 a0	197
6.26.3.3 alpha	197
6.26.3.4 alphas	197
6.26.3.5 calc	197
6.26.3.6 chl	197
6.26.3.7 chl0	198

6.26.3.8 chldyn	198
6.26.3.9 daylen	198
6.26.3.10 dlfa	198
6.26.3.11 dom	198
6.26.3.12 dynamicchl	198
6.26.3.13 f0n	199
6.26.3.14 fc	199
6.26.3.15 fcdtdq	199
6.26.3.16 ff	199
6.26.3.17 fn	199
6.26.3.18 fpic	199
6.26.3.19 ftalpha	200
6.26.3.20 ftemp	200
6.26.3.21 ftnf	200
6.26.3.22 ftrc	200
6.26.3.23 fv	200
6.26.3.24 growth	200
6.26.3.25 ic	201
6.26.3.26 ichl	201
6.26.3.27 idoc	201
6.26.3.28 ipic	201
6.26.3.29 mu0	201
6.26.3.30 n2f	201
6.26.3.31 ngr	201
6.26.3.32 nuts	202
6.26.3.33 pachl	202
6.26.3.34 par	202
6.26.3.35 pic	202
6.26.3.36 q0n	202
6.26.3.37 q0p	202
6.26.3.38 qpik	203
6.26.3.39 qsn	203
6.26.3.40 rc	203
6.26.3.41 rct	203
6.26.3.42 rctht	203
6.26.3.43 rdl	203
6.26.3.44 si	204
6.26.3.45 svph	204
6.26.3.46 tch	204
6.26.3.47 theta	204
6.26.3.48 v0	204
6.26.3.49 vc	204

6.26.3.50 vdic	205
6.26.3.51 vdoc	205
6.26.3.52 vn	205
6.26.3.53 vp	205
6.26.3.54 vph0	205
6.26.3.55 vpic	205
6.26.3.56 vsink	206
6.26.3.57 zc	206
6.26.3.58 zn	206
6.26.3.59 znf	206
6.26.3.60 znu	206
6.27 onf::phydat Type Reference	207
6.27.1 Detailed Description	208
6.27.2 Member Function/Subroutine Documentation	209
6.27.2.1 init()	209
6.27.3 Member Data Documentation	209
6.27.3.1 advect	209
6.27.3.2 botbc	209
6.27.3.3 bottom	209
6.27.3.4 cds	209
6.27.3.5 clsbot	210
6.27.3.6 data	210
6.27.3.7 fpar	210
6.27.3.8 get	210
6.27.3.9 ibot	210
6.27.3.10 isflx	210
6.27.3.11 pds	210
6.27.3.12 profile	211
6.27.3.13 sds	211
6.27.3.14 surface	211
6.27.3.15 surflux	211
6.27.3.16 surflx	211
6.27.3.17 surpar	211
6.28 et::preset Interface Reference	211
6.28.1 Detailed Description	212
6.28.2 Constructor & Destructor Documentation	212
6.28.2.1 preset()	212
6.29 onf::rpd Interface Reference	212
6.29.1 Detailed Description	212
6.29.2 Constructor & Destructor Documentation	212
6.29.2.1 rpd()	212
6.30 onf::statevars Type Reference	213

6.30.1 Detailed Description	214
6.30.2 Member Function/Subroutine Documentation	215
6.30.2.1 open()	215
6.30.3 Member Data Documentation	215
6.30.3.1 all_states	215
6.30.3.2 count	215
6.30.3.3 data	215
6.30.3.4 flux	216
6.30.3.5 flux_depth	216
6.30.3.6 ioflux	216
6.30.3.7 i0sms	216
6.30.3.8 it	216
6.30.3.9 nfl	216
6.30.3.10 nsms	216
6.30.3.11 nsv_all_boxes	217
6.30.3.12 nsv_one_box	217
6.30.3.13 nsv_total	217
6.30.3.14 roc	217
6.30.3.15 smsk	217
6.30.3.16 soma	217
6.30.3.17 start	217
6.30.3.18 svgn	218
6.30.3.19 var	218
6.30.3.20 write_output	218
6.30.3.21 write_soma	218
6.31 et::sunday Interface Reference	218
6.31.1 Detailed Description	218
6.31.2 Constructor & Destructor Documentation	218
6.31.2.1 sunday()	218
6.32 fudu::ut_decode_time Interface Reference	219
6.32.1 Detailed Description	219
6.32.2 Constructor & Destructor Documentation	219
6.32.2.1 ut_decode_time()	219
6.33 fudu::ut_encode_time Interface Reference	219
6.33.1 Detailed Description	219
6.33.2 Constructor & Destructor Documentation	220
6.33.2.1 ut_encode_time()	220
6.34 fudu::ut_free Interface Reference	220
6.34.1 Detailed Description	220
6.34.2 Constructor & Destructor Documentation	220
6.34.2.1 ut_free()	220
6.35 fudu::ut_free_system Interface Reference	220

6.35.1 Detailed Description	221
6.35.2 Constructor & Destructor Documentation	221
6.35.2.1 ut_free_system()	221
6.36 fudu::ut_get_converter Interface Reference	221
6.36.1 Detailed Description	221
6.36.2 Constructor & Destructor Documentation	221
6.36.2.1 ut_get_converter()	221
6.37 fudu::ut_get_status Interface Reference	221
6.37.1 Detailed Description	222
6.37.2 Constructor & Destructor Documentation	222
6.37.2.1 ut_get_status()	222
6.38 fudu::ut_parse Interface Reference	222
6.38.1 Detailed Description	222
6.38.2 Constructor & Destructor Documentation	222
6.38.2.1 ut_parse()	222
6.39 fudu::ut_read_xml Interface Reference	223
6.39.1 Detailed Description	223
6.39.2 Constructor & Destructor Documentation	223
6.39.2.1 ut_read_xml()	223
6.40 cfo::zoocfo Type Reference	223
6.40.1 Detailed Description	229
6.40.2 Member Function/Subroutine Documentation	229
6.40.2.1 flux()	229
6.40.2.2 read()	229
6.40.2.3 set()	230
6.40.3 Member Data Documentation	230
6.40.3.1 af	230
6.40.3.2 at	230
6.40.3.3 beta	230
6.40.3.4 ca	230
6.40.3.5 cf	230
6.40.3.6 cf0	231
6.40.3.7 cmax	231
6.40.3.8 cxda	231
6.40.3.9 cxdd	231
6.40.3.10 d0dvm	231
6.40.3.11 d0svm	231
6.40.3.12 depdvm	232
6.40.3.13 depsvm	232
6.40.3.14 dodvm	232
6.40.3.15 dodvm0	232
6.40.3.16 dom	232

6.40.3.17 dosvm	232
6.40.3.18 doya	233
6.40.3.19 doyd	233
6.40.3.20 dtdvm	233
6.40.3.21 dtsvma	233
6.40.3.22 dtsvmd	233
6.40.3.23 dvmig	233
6.40.3.24 dynsvm	234
6.40.3.25 e	234
6.40.3.26 egest	234
6.40.3.27 egestion	234
6.40.3.28 emax	234
6.40.3.29 eq	234
6.40.3.30 excrete	235
6.40.3.31 fdchl	235
6.40.3.32 fde	235
6.40.3.33 fdpic	235
6.40.3.34 fhphi	235
6.40.3.35 fhrm	235
6.40.3.36 forage	236
6.40.3.37 fpx	236
6.40.3.38 fsm	236
6.40.3.39 g	236
6.40.3.40 gmax	236
6.40.3.41 i	236
6.40.3.42 icdet	237
6.40.3.43 id0svm	237
6.40.3.44 id0svm1	237
6.40.3.45 id1dvm	237
6.40.3.46 id1svm	237
6.40.3.47 id1svm1	237
6.40.3.48 ida	238
6.40.3.49 idd	238
6.40.3.50 idet	238
6.40.3.51 idig	238
6.40.3.52 idim	238
6.40.3.53 idom	238
6.40.3.54 imaf	238
6.40.3.55 imax	239
6.40.3.56 imot	239
6.40.3.57 inot	239
6.40.3.58 ip	239

6.40.3.59 irda	239
6.40.3.60 irdd	239
6.40.3.61 isda	239
6.40.3.62 isdd	240
6.40.3.63 mort	240
6.40.3.64 ngr	240
6.40.3.65 pcc	240
6.40.3.66 phi	240
6.40.3.67 phi0	240
6.40.3.68 phim	241
6.40.3.69 phin	241
6.40.3.70 pp	241
6.40.3.71 ppm	241
6.40.3.72 ppn	241
6.40.3.73 pq	241
6.40.3.74 pth	242
6.40.3.75 qn_param	242
6.40.3.76 qp_param	242
6.40.3.77 quotas	242
6.40.3.78 r	242
6.40.3.79 rff	242
6.40.3.80 rffm	243
6.40.3.81 rffn	243
6.40.3.82 rm	243
6.40.3.83 rm0	243
6.40.3.84 rq	243
6.40.3.85 svmig	243
6.40.3.86 switch	244
6.40.3.87 thcsvm	244
6.40.3.88 toy	244
6.40.3.89 v0svm	244
6.40.3.90 vdvms	244
6.40.3.91 vsvm	244
6.40.3.92 vsvm0	245
6.40.3.93 vsvm1	245
6.40.3.94 xq	245

7 File Documentation 247

7.1 /Users/mpahlow/oppla/src/bac.f90 File Reference	247
7.2 bac.f90	248
7.3 /Users/mpahlow/oppla/src/brock81.f90 File Reference	251
7.4 brock81.f90	252

7.5 /Users/mpahlow/oppla/src/cfo.f90 File Reference	254
7.5.1 Data Type Documentation	255
7.5.1.1 type cfo::pcc	255
7.6 cfo.f90	255
7.7 /Users/mpahlow/oppla/src/clops.f90 File Reference	263
7.8 clops.f90	263
7.9 /Users/mpahlow/oppla/src/cmo.f90 File Reference	264
7.10 cmo.f90	265
7.11 /Users/mpahlow/oppla/src/det.f90 File Reference	270
7.12 det.f90	270
7.13 /Users/mpahlow/oppla/src/dic.f90 File Reference	273
7.13.1 Function/Subroutine Documentation	274
7.13.1.1 asf_o2()	274
7.13.1.2 deltaalk()	275
7.13.1.3 hset_m13()	275
7.14 dic.f90	277
7.15 /Users/mpahlow/oppla/src/dvode.f90 File Reference	284
7.16 dvode.f90	285
7.17 /Users/mpahlow/oppla/src/et.f90 File Reference	286
7.17.1 Data Type Documentation	288
7.17.1.1 type et::layer	288
7.17.1.2 type et::local	289
7.17.1.3 type et::state	291
7.17.1.4 type et::timing	291
7.18 et.f90	292
7.19 /Users/mpahlow/oppla/src/fudu.F90 File Reference	296
7.19.1 Data Type Documentation	297
7.19.1.1 type fudu::quantity	297
7.20 fudu.F90	297
7.21 /Users/mpahlow/oppla/src/julian.f90 File Reference	299
7.21.1 Data Type Documentation	301
7.21.1.1 type julian::leap	301
7.21.1.2 type julian::tm_struct	301
7.21.1.3 type julian::yms	302
7.22 julian.f90	302
7.23 /Users/mpahlow/oppla/src/lambert.f90 File Reference	306
7.24 lambert.f90	307
7.25 /Users/mpahlow/oppla/src/onf.f90 File Reference	312
7.25.1 Data Type Documentation	313
7.25.1.1 type onf::dataset	313
7.25.1.2 type onf::nf90stratt	313
7.25.2 Function/Subroutine Documentation	314

7.25.2.1 set_att()	314
7.26 onf.f90	314
7.27 /Users/mpahlow/oppla/src/oppla.F90 File Reference	324
7.27.1 Function/Subroutine Documentation	325
7.27.1.1 catch_signal_2()	325
7.27.1.2 oppla()	325
7.28 oppla.F90	326
7.29 /Users/mpahlow/oppla/src/plankton.f90 File Reference	328
7.29.1 Data Type Documentation	329
7.29.1.1 type plankton::bc	329
7.29.2 Function/Subroutine Documentation	330
7.29.2.1 expand_path()	330
7.29.2.2 lfn()	330
7.29.2.3 set_community()	330
7.29.2.4 set_environment()	331
7.29.2.5 set_timing()	331
7.30 plankton.f90	332
7.31 /Users/mpahlow/oppla/src/stdunt.f90 File Reference	342
7.32 stdunt.f90	342
Bibliography	343
Index	345

Chapter 1

Modules Index

1.1 Modules List

Here is a list of all modules with brief descriptions:

bac	Functions for bacteria and DOM	9
brock81	Functions for solar radiation and the diurnal light cycle	13
cfo	Optimal current-feeding model for zooplankton	23
clops	Process command-line options	32
cmo	Chain-model of optimal phytoplankton growth and diazotrophy	33
det	Detritus functions	41
dic	Functions for the carbonate system and alkalinity	43
dvoke	VODE_F90 interface for oppla	71
et	Ecological types	72
fudu	FORTRAN interface for the UDUNITS2 library	81
julian	Fortran implementation of the julian library	88
lambert	F90 module for the Lambert-W function made from TOMS 743	106
onf	NetCDF interface for oppla	108
plankton	Set up and drive marine ecosystem dynamics	120
stdunt	Common parameters for oppla (dp, stdin, stdout, stderr)	130

Chapter 2

Data Type Index

2.1 Class Hierarchy

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

et::aggregation	131
et::attenuate	131
plankton::bc	120
et::boxalk	140
et::boxbc	140
julian::c_strftime	141
julian::c_strptime	142
fudu::cv_convert	142
fudu::cv_free	143
onf::dataset	108
et::decl	144
dic::dicsys	151
onf::dimension	161
et::fluxes	167
et::fpar	167
et::fungroup	168
bac::bacteria	132
bac::dom	163
cfo::zoocfo	223
cmo::phycmo	191
det::detritus	144
et::init	175
lambert::lambertw	176
et::layer	72
julian::leap	88
et::light	177
et::local	72
onf::nf90group	179
onf::phydat	207
onf::statevars	213
onf::nf90stratt	108
onf::nf90var	183
onf::nf90grd	177
onf::nf90vec	185

dvode::ode	186
cfo::pcc	23
et::preset	211
fudu::quantity	81
onf::rpd	212
et::state	72
et::sunday	218
et::timing	72
julian::tm_struct	88
fudu::ut_decode_time	219
fudu::ut_encode_time	219
fudu::ut_free	220
fudu::ut_free_system	220
fudu::ut_get_converter	221
fudu::ut_get_status	221
fudu::ut_parse	222
fudu::ut_read_xml	223
julian::yms	88

Chapter 3

Data Type Index

3.1 Data Types List

Here are the data types with brief descriptions:

et::aggregation	Calculate aggregation fluxes	131
et::attenuate	Light attenuation	131
bac::bacteria	132
et::boxalk	140
et::boxbc	140
julian::c_strftime	Interface to POSIX strftime. Returns a formatted time string, given input time struct and format. See https://www.cplusplus.com/reference/ctime/strftime for reference	141
julian::c_strptime	Interface to POSIX strptime. Returns a time struct object based on the input time string str and format. See https://man7.org/linux/man-pages/man3/strptime.3.html for reference	142
fudu::cv_convert	142
fudu::cv_free	143
et::decl	Daylength function	144
det::detritus	144
dic::dicsys	151
onf::dimension	161
bac::dom	163
et::fluxes	Calculate fluxes in the source matrix soma	167
et::fpar	Fractions of surface PAR arriving at the box bottoms	167
et::fungroup	168
et::init	Read namelists	175
lambert::lambertw	176
et::light	Calculate or read surface irradiance	177
onf::nf90grd	177
onf::nf90group	179
onf::nf90var	183

onf::nf90vec	185
dcode::ode	186
cmo::phycmo	
Ordinary phytoplankton and diazotrophs	191
onf::phydat	
Physical data	207
et::preset	
Define ratios, set indices, pointers	211
onf::rpd	212
onf::statevars	213
et::sunday	218
fudu::ut_decode_time	219
fudu::ut_encode_time	219
fudu::ut_free	220
fudu::ut_free_system	220
fudu::ut_get_converter	221
fudu::ut_get_status	221
fudu::ut_parse	
Parse unit string	222
fudu::ut_read_xml	
Set-up and return units system from xml database	223
cfo::zoocfo	
Zooplankton	
Components depsvm, v0svm, dtsvma, and (doya and doyd) or svm must be set to enable sea-	
sonal vertical migration.	
Components depdvm and vdvdm must be set to enable diel vertical migration	223

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

/Users/mpahlow/oppla/src/ bac.f90	247
/Users/mpahlow/oppla/src/ brock81.f90	251
/Users/mpahlow/oppla/src/ cfo.f90	254
/Users/mpahlow/oppla/src/ clops.f90	263
/Users/mpahlow/oppla/src/ cmo.f90	264
/Users/mpahlow/oppla/src/ det.f90	270
/Users/mpahlow/oppla/src/ dic.f90	273
/Users/mpahlow/oppla/src/ dvoke.f90	284
/Users/mpahlow/oppla/src/ et.f90	286
/Users/mpahlow/oppla/src/ fudu.F90	296
/Users/mpahlow/oppla/src/ julian.f90	299
/Users/mpahlow/oppla/src/ lambert.f90	306
/Users/mpahlow/oppla/src/ onf.f90	312
/Users/mpahlow/oppla/src/ oppla.F90	324
/Users/mpahlow/oppla/src/ plankton.f90	328
/Users/mpahlow/oppla/src/ stdunt.f90	342

Chapter 5

Module Documentation

5.1 bac Module Reference

functions for bacteria and DOM

Data Types

- type [bacteria](#)
- type [dom](#)

Functions/Subroutines

- subroutine [bac_flux](#) (grp, grps, env, box, times)
- subroutine [bac_grow](#) (bac, box)
- subroutine [bac_read](#) (grp, env, lun)
- subroutine [bac_set](#) (grp, grps, env)
- real([dp](#)) function [bac_uptake](#) (bac)
saturation of DOC uptake (independent of temperature)
- real([dp](#)) function [bac_uptake_ft](#) (bac)
saturation of DOC uptake depends on temperature
- subroutine [dom_flux](#) (grp, grps, env, box, times)
set DOM-related fluxes in the source matrix soma
- subroutine [dom_read](#) (grp, env, lun)
read namelist dom and set dom parameters, constituents, number of states
- subroutine [dom_set](#) (grp, grps, env)
flag DOM-related elements of soma for output

5.1.1 Detailed Description

functions for bacteria and DOM

5.1.2 Function/Subroutine Documentation

5.1.2.1 bac_flux()

```
subroutine bac::bac_flux (
    class(bacteria), intent(inout) grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout) env,
    type(layer), dimension(0:), intent(inout) box,
    type(timing), intent(in) times ) [private]
```

Definition at line 146 of file [bac.f90](#).

5.1.2.2 bac_grow()

```
subroutine bac::bac_grow (
    class(bacteria), intent(inout) bac,
    type(layer), intent(in) box ) [private]
```

Definition at line 163 of file [bac.f90](#).

Referenced by [bac_read\(\)](#).

Here is the caller graph for this function:



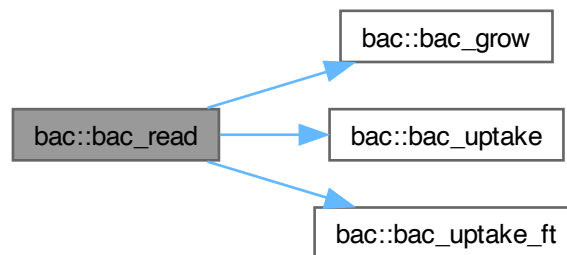
5.1.2.3 bac_read()

```
subroutine bac::bac_read (
    class(bacteria), intent(inout), target grp,
    type(local), intent(in), target env,
    integer, intent(in) lun )
```

Definition at line 47 of file [bac.f90](#).

References [bac_grow\(\)](#), [bac_uptake\(\)](#), and [bac_uptake_ft\(\)](#).

Here is the call graph for this function:



5.1.2.4 bac_set()

```

subroutine bac::bac_set (
    class(bacteria), intent(inout), target grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout), target env ) [private]
  
```

Definition at line 108 of file [bac.f90](#).

5.1.2.5 bac_uptake()

```

real(dp) function bac::bac_uptake (
    class(bacteria), intent(in) bac ) [private]
  
```

saturation of DOC uptake (independent of temperature)

Definition at line 206 of file [bac.f90](#).

Referenced by [bac_read\(\)](#).

Here is the caller graph for this function:



5.1.2.6 bac_uptake_ft()

```
real(dp) function bac::bac_uptake_ft (
    class(bacteria), intent(in) bac ) [private]
```

saturation of DOC uptake depends on temperature

Definition at line 197 of file [bac.f90](#).

Referenced by [bac_read\(\)](#).

Here is the caller graph for this function:



5.1.2.7 dom_flux()

```
subroutine bac::dom_flux (
    class(dom), intent(inout) grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout) env,
    type(layer), dimension(0:), intent(inout) box,
    type(timing), intent(in) times ) [private]
```

set DOM-related fluxes in the source matrix soma

Definition at line 263 of file [bac.f90](#).

5.1.2.8 dom_read()

```
subroutine bac::dom_read (
    class(dom), intent(inout), target grp,
    type(local), intent(in), target env,
    integer, intent(in) lun ) [private]
```

read namelist dom and set dom parameters, constituents, number of states

Definition at line 215 of file [bac.f90](#).

5.1.2.9 dom_set()

```
subroutine bac::dom_set (
    class(dom), intent(inout), target grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout), target env ) [private]
```

flag DOM-related elements of soma for output

Definition at line 253 of file [bac.f90](#).

5.2 brock81 Module Reference

Functions for solar radiation and the diurnal light cycle.

Functions/Subroutines

- subroutine, public `brock81_init` (envir, times)
Initialise the `brock81` module.
- real(dp) function `csrad` (times, hour)
clear-sky insolation at sea level
- subroutine `daylength` (times)
daylength in decimal days
- subroutine `declination_b81` (times)
solar declination after Brock (1981)
- subroutine `declination_f95` (times)
solar declination after Forsythe:1995
- subroutine `par_brock81` (ambient, time)
instantaneous surface irradiance
- subroutine `par_ld` (ambient, time)
- subroutine `par_mesoqua` (ambient, time)
- real(dp) function `rdrad` (times, hour)
ratio of actual to average daily irradiance
- subroutine `sunday_brock81` (times, env)
sunrise, noon, sunset and day length at times%now after Brock (1981)
- subroutine `sunday_const` (times, env)
- subroutine `sunday_ld` (times, env)
sunrise, noon, sunset for artificial LD cycle with constant daylength
- subroutine `sunday_mesoqua` (times, env)
sunrise, noon, sunset and daylength at timesnow for MesoAqua

Variables

- real(dp), parameter `cfhds` = 0.5_dp * cfdss
12h in s
- real(dp), parameter `d15` = rad * 15._dp
15° in rad
- real(dp), parameter `fd` = rad * 360._dp / yearDays
conversion factor from doy to rad
- real(dp) `sdl`
*short form for SIN(d1)*SIN(latrad) (+ SIN(ptl))*
- real(dp), parameter `solar` = 1.353E3_dp
solar constant in W/m²
- real(dp), parameter `tilt` = rad * 23.45_dp
tilt of the earth against its orbital plane in radians
- real(dp), parameter `yeardays` = 365.2425_dp
length of a year in days

5.2.1 Detailed Description

Functions for solar radiation and the diurnal light cycle.

The diurnal light cycle is described without twilight after [Brook \(1981\)](#) and with twilight after [Forsythe et al. \(1995\)](#).

5.2.2 Function/Subroutine Documentation

5.2.2.1 brock81_init()

```
subroutine, public brock81::brock81_init (
    type(local), intent(inout) envir,
    type(timing), intent(inout), target times )
```

Initialise the [brock81](#) module.

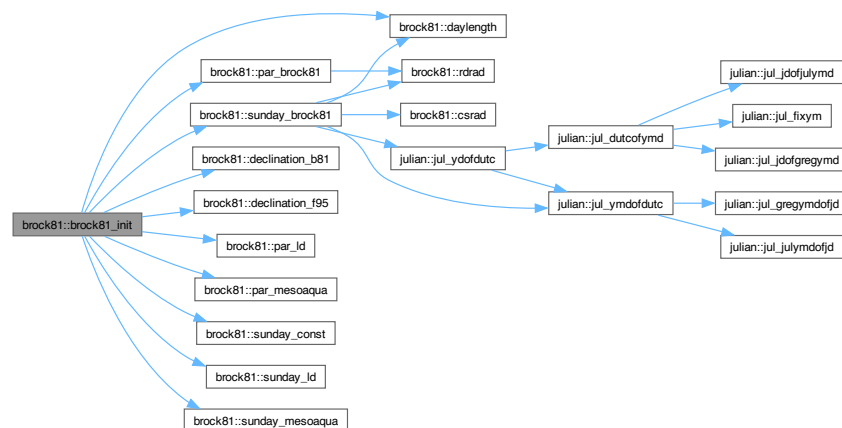
Parameters

in	<i>envir%daylenfun</i>	how to determine solar declination: 'Brock81' is without, 'Forsythe95' with twilight
in	<i>envir%dicy</i>	diurnal light cycle: " or 'none' means constant light, 'Brock81' natural light cycle, 'MesoAqua' light cycle for MesoAqua experiments, 'LD' fixed rectangular light cycle

Definition at line 25 of file [brock81.f90](#).

References [daylength\(\)](#), [declination_b81\(\)](#), [declination_f95\(\)](#), [par_brock81\(\)](#), [par_ld\(\)](#), [par_mesoqua\(\)](#), [et::rad](#), [sunday_brock81\(\)](#), [sunday_const\(\)](#), [sunday_ld\(\)](#), and [sunday_mesoqua\(\)](#).

Here is the call graph for this function:



5.2.2.2 csrad()

```
real(dp) function brock81::csrad (
    class(timing), intent(in) times,
    real(dp), intent(in) hour ) [private]
```


clear-sky insolation at sea level

The difference between the functions for [Brock \(1981\)](#) and [Forsythe et al. \(1995\)](#) is stored in `sdl` and `times%d1`.

Parameters

in	<code>times%doy</code>	day of year
in	<code>times%latrad</code>	latitude N in rad
in	<code>times%d1</code>	solar declination
in	<code>hour</code>	time of day in decimal hours local solar time
in	<code>sdl</code>	$\text{SIN}(\text{times}\% \text{latrad}) * \text{SIN}(\text{times}\% \text{d1})$

Returns

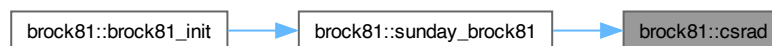
insolation in W m^{-2}

Definition at line 162 of file [brock81.f90](#).

References [d15](#), [fd](#), [sdl](#), and [solar](#).

Referenced by [sunday_brock81\(\)](#).

Here is the caller graph for this function:



5.2.2.3 daylength()

```

subroutine brock81::daylength (
    class(timing), intent(inout) times ) [private]

```

daylength in decimal days

Parameters

in	<code>times%latrad</code>	latitude N in rad
in	<code>times%doy</code>	day of year
out	<code>times%daylen</code>	daylength decimal fraction

Returns

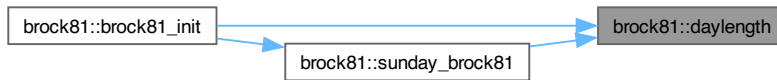
daylength (light period) as a fraction of 24 h

Definition at line 238 of file [brock81.f90](#).

References [et::pi](#), and [sdl](#).

Referenced by [brock81_init\(\)](#), and [sunday_brock81\(\)](#).

Here is the caller graph for this function:



5.2.2.4 declination_b81()

```
subroutine brock81::declination_b81 (
    class(timing), intent(inout) times ) [private]
```

solar declination after [Brook \(1981\)](#)

Parameters

in	<i>times%latrad</i>	latitude N in rad
in	<i>times%doy</i>	day of year
out	<i>times%d1</i>	declination
out	<i>sdl</i>	$\text{SIN}(\text{times\%latrad}) * \text{SIN}(\text{times\%d1})$

Definition at line 211 of file [brock81.f90](#).

References [fd](#), [sdl](#), and [tilt](#).

Referenced by [brock81_init\(\)](#).

Here is the caller graph for this function:



5.2.2.5 declination_f95()

```
subroutine brock81::declination_f95 (
    class(timing), intent(inout) times ) [private]
```

solar declination after Forsythe:1995

Parameters

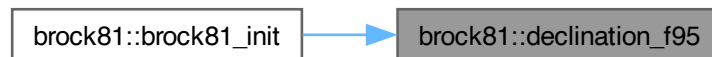
in	<i>times%latrad</i>	latitude N in rad
in	<i>times%doy</i>	day of year
in	<i>times%ptl</i>	twilight parameter in rad
out	<i>times%d1</i>	declination
out	<i>sdl</i>	$\text{SIN}(\text{times}\%ptl) + \text{SIN}(\text{times}\%latrad) * \text{SIN}(\text{times}\%d1)$

Definition at line 224 of file [brock81.f90](#).

References [et::pi](#), [sdl](#), and [yeardays](#).

Referenced by [brock81_init\(\)](#).

Here is the caller graph for this function:



5.2.2.6 par_brock81()

```

subroutine brock81::par_brock81 (
    class(local), intent(inout) ambient,
    type(timing), intent(inout) time ) [private]
  
```

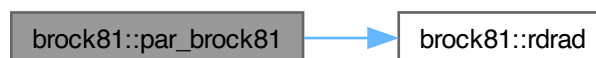
instantaneous surface irradiance

Definition at line 96 of file [brock81.f90](#).

References [rdrad\(\)](#).

Referenced by [brock81_init\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.7 `par_ld()`

```
subroutine brock81::par_ld (  
    class(local), intent(inout) ambient,  
    type(timing), intent(inout) time ) [private]
```

Definition at line 135 of file `brock81.f90`.

Referenced by `brock81_init()`.

Here is the caller graph for this function:



5.2.2.8 `par_mesoqua()`

```
subroutine brock81::par_mesoqua (  
    class(local), intent(inout) ambient,  
    type(timing), intent(inout) time ) [private]
```

Definition at line 115 of file `brock81.f90`.

Referenced by `brock81_init()`.

Here is the caller graph for this function:



5.2.2.9 rdrad()

```
real(dp) function brock81::rdrad (
    class(timing), intent(in) times,
    real(dp), intent(in) hour ) [private]
```

ratio of actual to average daily irradiance

Equations 14-16 in Brock (1981) don't seem to work; thus, dr is calculated here from the ratio of Eq. 26 over Eq. 10 (without the factor 24). The difference between the functions for [Brock \(1981\)](#) and [Forsythe et al. \(1995\)](#) is stored in sdl

Parameters

in	<i>times%day</i>	day of the year
in	<i>times%latrad</i>	latitude N in rad
in	<i>hour</i>	time of day in decimal hours

Returns

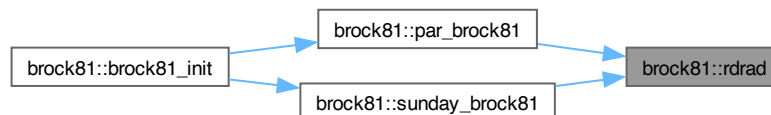
irradiance ratio

Definition at line 192 of file [brock81.f90](#).

References [d15](#), [et::pi](#), and [sdl](#).

Referenced by [par_brock81\(\)](#), and [sunday_brock81\(\)](#).

Here is the caller graph for this function:



5.2.2.10 sunday_brock81()

```
subroutine brock81::sunday_brock81 (
    class(timing), intent(inout) times,
    type(local), intent(inout) env ) [private]
```

sunrise, noon, sunset and day length at times%now after [Brock \(1981\)](#)

Parameters

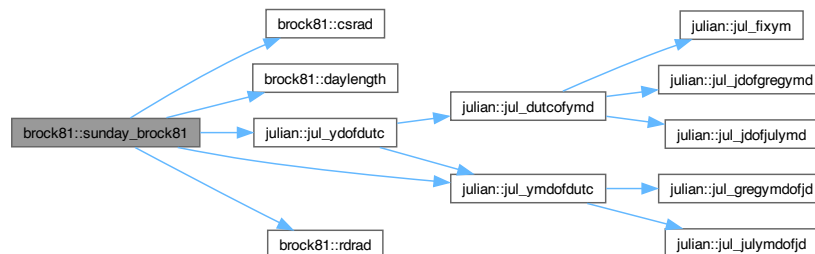
in	<i>times%now</i>	current time in s
in	<i>times%latrad</i>	latitude N in radians
out	<i>times%daylen</i>	current daylength in d
Generated by Doxygen	<i>times%sun</i>	closest next sunrise, noon, sunset in UTC

Definition at line 68 of file [brock81.f90](#).

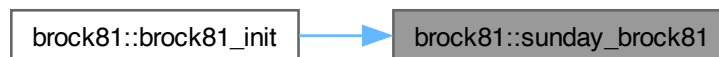
References [csrad\(\)](#), [daylength\(\)](#), [julian::jul_ydofdutc\(\)](#), [julian::jul_ymdofdutc\(\)](#), and [rdrad\(\)](#).

Referenced by [brock81_init\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.11 `sunday_const()`

```

subroutine brock81::sunday_const (
    class(timing), intent(inout) times,
    type(local), intent(inout) env ) [private]

```

Definition at line 146 of file [brock81.f90](#).

Referenced by [brock81_init\(\)](#).

Here is the caller graph for this function:



5.2.2.12 `sunday_ld()`

```
subroutine brock81::sunday_ld (
    class(timing), intent(inout) times,
    type(local), intent(inout) env ) [private]
```

sunrise, noon, sunset for artificial LD cycle with constant daylength

Parameters

in	<i>times%now</i>	current time in s
in	<i>env%daylen</i>	daylength in d
out	<i>times%sun</i>	latitude N in radians

Definition at line 127 of file `brock81.f90`.

Referenced by `brock81_init()`.

Here is the caller graph for this function:



5.2.2.13 `sunday_mesoqua()`

```
subroutine brock81::sunday_mesoqua (
    class(timing), intent(inout) times,
    type(local), intent(inout) env ) [private]
```

sunrise, noon, sunset and daylength at timesnow for MesoAqua

Light cycle of the MesoAqua experiments by [Lewandowska and Sommer \(2010\)](#)

Definition at line 107 of file `brock81.f90`.

Referenced by `brock81_init()`.

Here is the caller graph for this function:



5.2.3 Variable Documentation

5.2.3.1 cfhds

```
real(dp), parameter brock81::cfhds =0.5_dp*cfds [private]
```

12h in s

Definition at line 11 of file [brock81.f90](#).

5.2.3.2 d15

```
real(dp), parameter brock81::d15 =rad*15._dp [private]
```

15° in rad

Definition at line 11 of file [brock81.f90](#).

Referenced by [csrad\(\)](#), and [rdrad\(\)](#).

5.2.3.3 fd

```
real(dp), parameter brock81::fd =rad*360._dp/yearDays [private]
```

conversion factor from day to rad

Definition at line 11 of file [brock81.f90](#).

Referenced by [csrad\(\)](#), and [declination_b81\(\)](#).

5.2.3.4 sdl

```
real(dp) brock81::sdl [private]
```

short form for SIN(d1)*SIN(latrad) (+ SIN(ptl))

Definition at line 17 of file [brock81.f90](#).

Referenced by [csrad\(\)](#), [daylength\(\)](#), [declination_b81\(\)](#), [declination_f95\(\)](#), and [rdrad\(\)](#).

5.2.3.5 solar

```
real(dp), parameter brock81::solar =1.353E3_dp [private]
```

solar constant in W/m²

Definition at line 11 of file [brock81.f90](#).

Referenced by [csrad\(\)](#).

5.2.3.6 tilt

```
real(dp), parameter brock81::tilt =rad*23.45_dp [private]
```

tilt of the earth against its orbital plane in radians

Definition at line 11 of file [brock81.f90](#).

Referenced by [declination_b81\(\)](#).

5.2.3.7 yeardays

```
real(dp), parameter brock81::yeardays =365.2425_dp [private]
```

length of a year in days

Definition at line 11 of file [brock81.f90](#).

Referenced by [declination_f95\(\)](#).

5.3 cfo Module Reference

Optimal current-feeding model for zooplankton.

Data Types

- type [pcc](#)
Associate prey type with prey capture coefficient. [More...](#)
- type [zoocfo](#)
*Zooplankton
Components `depsvm`, `v0svm`, `dtsvm`, and (`doya` and `doyd`) or `svm` must be set to enable seasonal vertical migration.
Components `depdvm` and `vdvm` must be set to enable diel vertical migration.*

Functions/Subroutines

- subroutine [cfo_dvm](#) (zoo, box, times)
Vertical velocity for diel vertical migration.
- subroutine [cfo_flux](#) (grp, grps, env, box, times)
flux routine for the cfo module
- subroutine [cfo_read](#) (grp, env, lun)
Read namelist cfo for matching species.
- subroutine [cfo_set](#) (grp, grps, env)
- subroutine [cfo_svm_dyn](#) (zoo, box, times)
Vertical velocity for seasonal vertical migration and days of ascent and descent.
- subroutine [cfo_svm_static](#) (zoo, box, times)
Vertical velocities and days of ascent and descent.
- subroutine [egest_dm](#) (zoo, box)
- subroutine [egest_pm](#) (zoo, box)
- subroutine [excrete_dim](#) (zoo, env, box)
- subroutine [excrete_dom](#) (zoo, env, box)
- subroutine [foract](#) (zoo, box, pc, ei, rm)
- subroutine [forage_cfo](#) (zoo, env, box)
- subroutine [forage_switch](#) (zoo, env, box)

5.3.1 Detailed Description

Optimal current-feeding model for zooplankton.

This module implements the model described by [Pahlow and Prowe \(2010\)](#). Zooplankton groups can feed on different types of prey identified by their species names. Different zooplankton groups can also be distinguished by their vertical migration behaviour.

5.3.2 Data Type Documentation

5.3.2.1 type cfo::pcc

Associate prey type with prey capture coefficient.

Definition at line 13 of file [cfo.f90](#).

Class Members

type(quantity)	phi	prey capture coefficient
character(len=100)	type	prey type, corresponds to groupname

5.3.3 Function/Subroutine Documentation

5.3.3.1 cfo_dvm()

```
subroutine cfo::cfo_dvm (
    class(zoo_cfo), intent(inout) zoo,
    type(layer), intent(in) box,
    type(timing), intent(in) times ) [private]
```

Vertical velocity for diel vertical migration.

Parameters

in	<i>zoo%id1dvm</i>	index of day-time level
in	<i>zoo%d0dvm</i>	night-time depth of diel vertical migration
in	<i>zoo%depdvm</i>	day-time target depth of diel vertical migration
in	<i>zoo%dt_dvm</i>	time window of diel vertical migration
in	<i>box%depth</i>	depth of current layer
in	<i>times%current</i>	current time
in	<i>times%sunrise</i>	sunrise of current day in s
in	<i>times%sunset</i>	sunset of current day in s
out	<i>zoo%vv</i>	vertical velocity in m s^{-1}

Definition at line 419 of file [cfo.f90](#).

Referenced by [cfo_read\(\)](#).

Here is the caller graph for this function:



5.3.3.2 cfo_flux()

```

subroutine cfo::cfo_flux (
    class(zoo_cfo), intent(inout) grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout) env,
    type(layer), dimension(0:), intent(inout) box,
    type(timing), intent(in) times ) [private]
  
```

flux routine for the cfo module

Definition at line 377 of file [cfo.f90](#).

5.3.3.3 cfo_read()

```

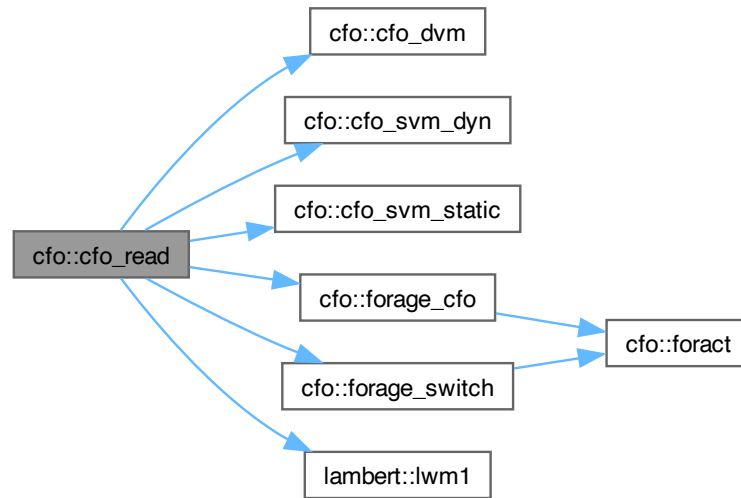
subroutine cfo::cfo_read (
    class(zoo_cfo), intent(inout), target grp,
    type(local), intent(in), target env,
    integer, intent(in) lun )
  
```

Read namelist cfo for matching species.

Definition at line 114 of file [cfo.f90](#).

References [cfo_dvm\(\)](#), [cfo_svm_dyn\(\)](#), [cfo_svm_static\(\)](#), [forage_cfo\(\)](#), [forage_switch\(\)](#), and [lambert::lwm1\(\)](#).

Here is the call graph for this function:



5.3.3.4 cfo_set()

```

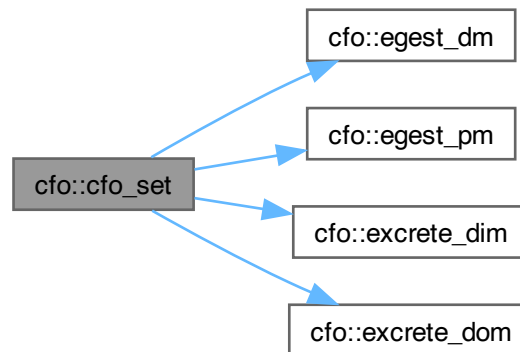
subroutine cfo::cfo_set (
    class(zoo_cfo), intent(inout), target grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout), target env ) [private]

```

Definition at line 287 of file `cfo.f90`.

References `egest_dm()`, `egest_pm()`, `excrete_dim()`, and `excrete_dom()`.

Here is the call graph for this function:



5.3.3.5 cfo_svm_dyn()

```
subroutine cfo::cfo_svm_dyn (
    class(zoo_cfo), intent(inout) zoo,
    type(layer), dimension(0:), intent(in) box,
    type(timing), intent(in) times ) [private]
```

Vertical velocity for seasonal vertical migration and days of ascent and descent.

Subroutine `cfo_svm_dyn` implements the days of ascent and descent (`doya` and `doyd`) as dynamic traits. Animals leaving their target layer (`depsvm` for the ascent and `d0svm` for the descent) take the current day with them as the corresponding trait. This is implemented here by adding a correction to the corresponding source term in the adjacent layer in the direction of movement.

Parameters

in	<code>zoo%dtsvma</code>	width of time window for seasonal vertical ascent
in	<code>zoo%dtsvmd</code>	width of time window for seasonal vertical descent
in	<code>zoo%depsvm</code>	winter depth for seasonal vertical migration
in	<code>zoo%d0svm</code>	summer depth for seasonal vertical migration
in	<code>box%depth</code>	mid-depth of current layer
in	<code>box%height</code>	height of current layer

Definition at line 452 of file `cfo.f90`.

Referenced by `cfo_read()`.

Here is the caller graph for this function:



5.3.3.6 cfo_svm_static()

```
subroutine cfo::cfo_svm_static (
    class(zoo_cfo), intent(inout) zoo,
    type(layer), dimension(0:), intent(in) box,
    type(timing), intent(in) times ) [private]
```

Vertical velocities and days of ascent and descent.

Subroutine `cfo_svm_static` is for static days of ascent and descent, i.e., `doya` and `doyd` are parameters in the corresponding `cfo` namelist.

Parameters

in	<i>zoo%doya</i>	day of year of ascent
in	<i>zoo%doyd</i>	day of year of descent
in	<i>zoo%dtsvma</i>	width of time window for seasonal vertical ascent
in	<i>zoo%dtsvmd</i>	width of time window for seasonal vertical descent
in	<i>zoo%depsvm</i>	winter depth for seasonal vertical migration
in	<i>zoo%d0svm</i>	summer depth for seasonal vertical migration
in	<i>box%depth</i>	mid-depth of current layer
in	<i>box%height</i>	height of current layer

Definition at line 531 of file [cfo.f90](#).

Referenced by [cfo_read\(\)](#).

Here is the caller graph for this function:



5.3.3.7 egest_dm()

```

subroutine cfo::egest_dm (
    class(zoocfo), intent(in) zoo,
    type(layer), intent(inout) box ) [private]
  
```

Definition at line 584 of file [cfo.f90](#).

Referenced by [cfo_set\(\)](#).

Here is the caller graph for this function:



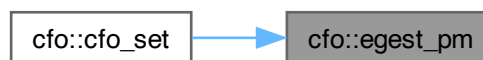
5.3.3.8 egest_pm()

```
subroutine cfo::egest_pm (  
    class(zoo_cfo), intent(in) zoo,  
    type(layer), intent(inout) box ) [private]
```

Definition at line 592 of file [cfo.f90](#).

Referenced by [cfo_set\(\)](#).

Here is the caller graph for this function:



5.3.3.9 excrete_dim()

```
subroutine cfo::excrete_dim (  
    class(zoo_cfo), intent(inout) zoo,  
    type(local), intent(inout) env,  
    type(layer), intent(inout) box ) [private]
```

Definition at line 575 of file [cfo.f90](#).

Referenced by [cfo_set\(\)](#).

Here is the caller graph for this function:



5.3.3.10 excrete_dom()

```
subroutine cfo::excrete_dom (
    class(zoo_cfo), intent(inout) zoo,
    type(local), intent(inout) env,
    type(layer), intent(inout) box ) [private]
```

Definition at line 561 of file [cfo.f90](#).

Referenced by [cfo_set\(\)](#).

Here is the caller graph for this function:



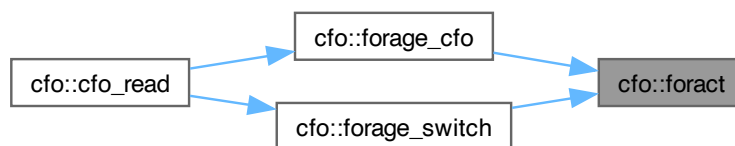
5.3.3.11 foract()

```
subroutine cfo::foract (
    class(zoo_cfo), intent(inout) zoo,
    type(layer), intent(in) box,
    real(dp), intent(in) pc,
    real(dp), intent(out) ei,
    real(dp), intent(out) rm ) [private]
```

Definition at line 677 of file [cfo.f90](#).

Referenced by [forage_cfo\(\)](#), and [forage_switch\(\)](#).

Here is the caller graph for this function:



5.3.3.12 forage_cfo()

```
subroutine cfo::forage_cfo (
    class(zoo_cfo), intent(inout) zoo,
    type(local), intent(in) env,
    type(layer), intent(in) box ) [private]
```


Parameters

<code>in, out</code>	<code>zoo</code>	functional type
<code>in</code>	<code>env</code>	global environment
<code>in</code>	<code>box</code>	local environment

Definition at line 601 of file [cfo.f90](#).

References [foract\(\)](#).

Referenced by [cfo_read\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.13 forage_switch()

```

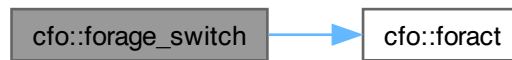
subroutine cfo::forage_switch (
    class(zoo_cfo), intent(inout) zoo,
    type(local), intent(in) env,
    type(layer), intent(in) box ) [private]
  
```

Definition at line 635 of file [cfo.f90](#).

References [foract\(\)](#).

Referenced by [cfo_read\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.4 clops Module Reference

process command-line options

Functions/Subroutines

- subroutine `getopt` (opts, args, flags, usage, help)
sort arguments according to an option string

5.4.1 Detailed Description

process command-line options

5.4.2 Function/Subroutine Documentation

5.4.2.1 getopt()

```

subroutine clops::getopt (
    character(len=*), intent(in)  opts,
    character(len=*), dimension(:), intent(inout) args,
    logical, dimension(:), intent(out) flags,
    character(len=*), intent(in)  usage,
    character(len=*), intent(in), optional help )
  
```

sort arguments according to an option string

option	meaning
-<x>	option <x> with argument or switch <x>
-?	prints a usage message
--	signals the end of options; arguments without option letters are left unsorted

Parameters

in	<i>opts</i>	string with option letters; a : must be included to signal the start of switches, if left out, all letters are switches
in, out	<i>args</i>	string array to hold arguments; the first n elements will be the arguments to the options in opts, with n = length of opts
out	<i>flags</i>	logical array to indicate which switches are present
in	<i>usage</i>	message to display for invalid input
in	<i>help</i>	help message displayed for -?

Definition at line 20 of file [clops.f90](#).

Referenced by [oppla\(\)](#).

Here is the caller graph for this function:



5.5 cmo Module Reference

Chain-model of optimal phytoplankton growth and diazotrophy.

Data Types

- type [phycmo](#)
Ordinary phytoplankton and diazotrophs.

Functions/Subroutines

- subroutine [chl_dyn](#) (phy, box)
- subroutine [cmo_flux](#) (grp, grps, env, box, times)
- subroutine [cmo_pic](#) (phy, box)
- subroutine [cmo_read](#) (grp, env, lun)
Read parameters for optimality-based chain model.
- subroutine [cmo_set](#) (grp, grps, env)

- subroutine [facn2f](#) (phy, box, v0)
- real(dp) function [ftnf_eppley](#) (phy, box)
- real(dp) function [ftnf_houlton](#) (phy, box)
- real(dp) function [ftnf_oppla](#) (phy, box)
- subroutine [grow](#) (phy, ambient, box)
 - Rates of growth, nutrient uptake, and chlorophyll synthesis or steady-state Chl:C ratio.*
- subroutine [tchdyn](#) (phy, ambient, box)
 - Dynamic photo-acclimation via Chl synthesis.*
- subroutine [tchia](#) (phy, ambient, box)
 - Instantaneous photo-acclimation to daytime-average light intensity.*
- subroutine [uptake](#) (phy, box, v0)

5.5.1 Detailed Description

Chain-model of optimal phytoplankton growth and diazotrophy.

This module implements the model described by [Pahlow et al. \(2013\)](#).

5.5.2 Function/Subroutine Documentation

5.5.2.1 chl_dyn()

```
subroutine cmo::chl_dyn (
    class(phycmo), intent(inout) phy,
    type(layer), intent(in) box ) [private]
```

Definition at line 378 of file [cmo.f90](#).

Referenced by [cmo_read\(\)](#).

Here is the caller graph for this function:



5.5.2.2 cmo_flux()

```
subroutine cmo::cmo_flux (
    class(phycmo), intent(inout) grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout) env,
    type(layer), dimension(0:), intent(inout) box,
    type(timing), intent(in) times ) [private]
```

Definition at line 258 of file [cmo.f90](#).

5.5.2.3 cmo_pic()

```
subroutine cmo::cmo_pic (
    class(phycmo), intent(inout) phy,
    type(layer), intent(inout) box ) [private]
```

Definition at line 436 of file [cmo.f90](#).

Referenced by [cmo_set\(\)](#).

Here is the caller graph for this function:



5.5.2.4 cmo_read()

```
subroutine cmo::cmo_read (
    class(phycmo), intent(inout), target grp,
    type(local), intent(in), target env,
    integer, intent(in) lun )
```

Read parameters for optimality-based chain model.

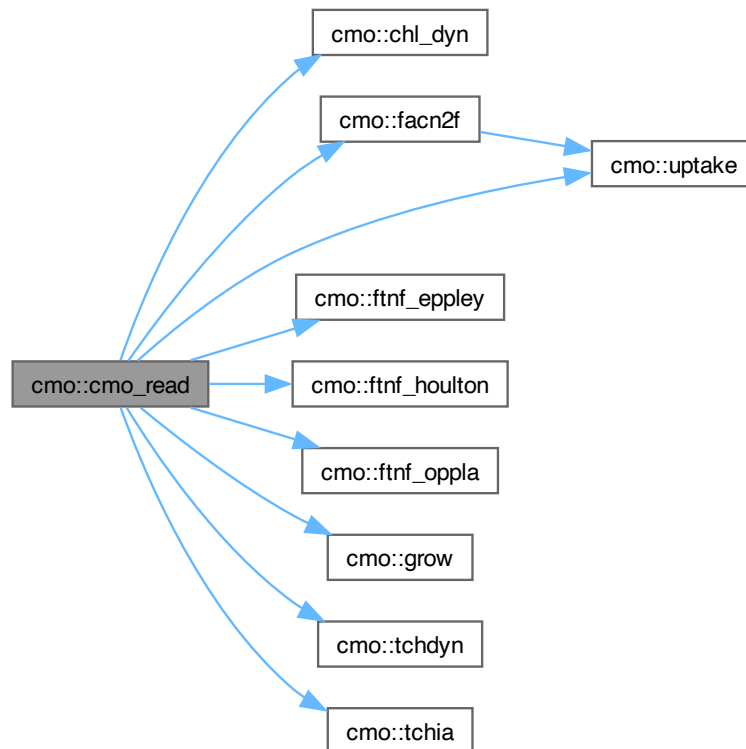
Parameters

in, out	<i>grp</i>	funcional type
in	<i>env</i>	local environment
in	<i>lun</i>	logical unit number for namelist cmo

Definition at line 78 of file [cmo.f90](#).

References [chl_dyn\(\)](#), [facn2f\(\)](#), [ftnf_eppley\(\)](#), [ftnf_houlton\(\)](#), [ftnf_oppla\(\)](#), [grow\(\)](#), [tchdyn\(\)](#), [tchia\(\)](#), and [uptake\(\)](#).

Here is the call graph for this function:



5.5.2.5 cmo_set()

```

subroutine cmo::cmo_set (
    class(phycmo), intent(inout), target grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout), target env ) [private]

```

Definition at line 209 of file [cmo.f90](#).

References [cmo_pic\(\)](#).

Here is the call graph for this function:



5.5.2.6 facn2f()

```
subroutine cmo::facn2f (  
    class(phycmo), intent(inout) phy,  
    type(layer), intent(in) box,  
    real(dp), intent(in) v0 ) [private]
```

Definition at line 355 of file [cmo.f90](#).

References [uptake\(\)](#).

Referenced by [cmo_read\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.2.7 ftnf_eppley()

```
real(dp) function cmo::ftnf_eppley (  
    class(phycmo), intent(in) phy,  
    type(layer), intent(in) box ) [private]
```

Definition at line 454 of file [cmo.f90](#).

Referenced by [cmo_read\(\)](#).

Here is the caller graph for this function:



5.5.2.8 ftnf_houlton()

```
real(dp) function cmo::ftnf_houlton (
    class(phycmo), intent(in) phy,
    type(layer), intent(in) box ) [private]
```

Definition at line 462 of file [cmo.f90](#).

Referenced by [cmo_read\(\)](#).

Here is the caller graph for this function:



5.5.2.9 ftnf_oppla()

```
real(dp) function cmo::ftnf_oppla (
    class(phycmo), intent(in) phy,
    type(layer), intent(in) box ) [private]
```

Definition at line 446 of file [cmo.f90](#).

Referenced by [cmo_read\(\)](#).

Here is the caller graph for this function:



5.5.2.10 grow()

```
subroutine cmo::grow (
    class(phycmo), intent(inout) phy,
    type(local), intent(in) ambient,
    type(layer), intent(in) box ) [private]
```

Rates of growth, nutrient uptake, and chlorophyll synthesis or steady-state Chl:C ratio.

Parameters

in	<i>ambient%daylen</i>	daylength as a fraction of 24 h
in	<i>box%DIN</i>	DIN concentration / mmol m^{-3}
in	<i>box%DIP</i>	DIP concentration / mmol m^{-3}
in	<i>box%dPAR</i>	average daytime PAR
in	<i>box%bPAR</i>	PAR at bottom of box (layer)
in	<i>box%tPAR</i>	PAR at top of box (layer)
in	<i>phy%QN</i>	N cell quota (N:C ratio)
in	<i>phy%QP</i>	P cell quota (P:C ratio)
in	<i>phy%A0</i>	potential nutrient affinity
in	<i>phy%alpha</i>	potential light affinity (initial slope)
in	<i>phy%QsN</i>	non-allocatable N quota ($0.5 \cdot \text{phy}Q0N$)
in	<i>phy%Q0N</i>	subsistence N quota (minimum N:C ratio)
in	<i>phy%Q0P</i>	subsistence P quota (minimum P:C ratio)
in	<i>phy%RC</i>	cost of Chl maintenance
in	<i>phy%rdl</i>	daylength parameter
in	<i>phy%V0</i>	maximum-rate parameter
in	<i>phy%zC</i>	cost of Chl synthesis
in	<i>phy%zN</i>	cost of biosynthesis
in	<i>phy%fT</i>	temperature function for growth
out	<i>phy%A</i>	light-dependence of growth
out	<i>phy%fC</i>	allocation factor for CO_2 fixation
out	<i>phy%fN</i>	allocation factor for N uptake
out	<i>phy%fV</i>	allocation factor for nutrient uptake
out	<i>phy%mu0</i>	potential gross CO_2 fixation
out	<i>phy%SI</i>	light saturation
out	<i>phy%rctht</i>	rate of change of the Chl:C ratio
out	<i>phy%tch</i>	chloroplast Chl:C ratio
out	<i>phy%ngr</i>	net growth rate (rate of C assimilation)
out	<i>phy%VN</i>	rate of N uptake
out	<i>phy%VP</i>	rate of P uptake
out	<i>phy%VNh0</i>	local potential N uptake
out	<i>phy%VPh0</i>	local potential P uptake

Definition at line 312 of file [cmo.f90](#).

Referenced by [cmo_read\(\)](#).

Here is the caller graph for this function:



5.5.2.11 tchdyn()

```
subroutine cmo::tchdyn (
    class(phycmo), intent(inout) phy,
    type(local), intent(in) ambient,
    type(layer), intent(in) box ) [private]
```

Dynamic photo-acclimation via Chl synthesis.

Parameters

in, out	<i>phy</i>	current phytoplankton group
in	<i>ambient</i>	ambient environment
in	<i>box</i>	current layer environment

Definition at line 392 of file [cmo.f90](#).

Referenced by [cmo_read\(\)](#).

Here is the caller graph for this function:



5.5.2.12 tchia()

```
subroutine cmo::tchia (
    class(phycmo), intent(inout) phy,
    type(local), intent(in) ambient,
    type(layer), intent(in) box ) [private]
```

Instantaneous photo-acclimation to daytime-average light intensity.

Definition at line 416 of file [cmo.f90](#).

Referenced by [cmo_read\(\)](#).

Here is the caller graph for this function:



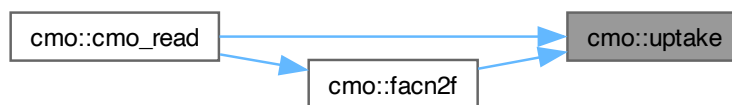
5.5.2.13 uptake()

```
subroutine cmo::uptake (
    class(phycmo), intent(inout) phy,
    type(layer), intent(in) box,
    real(dp), intent(in) v0 ) [private]
```

Definition at line 339 of file [cmo.f90](#).

Referenced by [cmo_read\(\)](#), and [facn2f\(\)](#).

Here is the caller graph for this function:



5.6 det Module Reference

detritus functions

Data Types

- type [detritus](#)

Functions/Subroutines

- subroutine [det_flux](#) (grp, grps, env, box, times)
- subroutine [det_fluxes](#) (grp, box)
- subroutine [det_loss_pic](#) (grp, box)
- subroutine [det_read](#) (grp, env, lun)

read detritus-related parameters, set detritus parameters, constituents, number of states
- subroutine [det_set](#) (grp, grps, env)

flag detritus fluxes in soma for output, set-up remineralisation scheme

5.6.1 Detailed Description

detritus functions

5.6.2 Function/Subroutine Documentation

5.6.2.1 `det_flux()`

```
subroutine det::det_flux (
    class(detritus), intent(inout) grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout) env,
    type(layer), dimension(0:), intent(inout) box,
    type(timing), intent(in) times ) [private]
```

Definition at line 160 of file `det.f90`.

5.6.2.2 `det_fluxes()`

```
subroutine det::det_fluxes (
    class(detritus), intent(in) grp,
    type(layer), intent(inout) box ) [private]
```

Definition at line 181 of file `det.f90`.

Referenced by `det_set()`.

Here is the caller graph for this function:



5.6.2.3 `det_loss_pic()`

```
subroutine det::det_loss_pic (
    class(detritus), intent(inout) grp,
    type(layer), intent(in) box ) [private]
```

Definition at line 172 of file `det.f90`.

Referenced by `det_set()`.

Here is the caller graph for this function:



5.6.2.4 det_read()

```
subroutine det::det_read (
    class(detritus), intent(inout), target grp,
    type(local), intent(in), target env,
    integer, intent(in) lun )
```

read detritus-related parameters, set detritus parameters, constituents, number of states

Parameters

in, out	<i>grp</i>	funcional type
in	<i>env</i>	local environment
in	<i>lun</i>	unit number of open file with namelist det

Definition at line 35 of file [det.f90](#).

5.6.2.5 det_set()

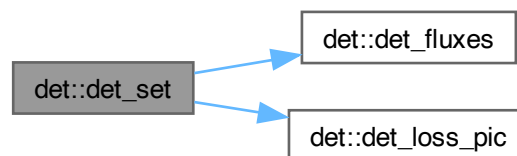
```
subroutine det::det_set (
    class(detritus), intent(inout), target grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout), target env ) [private]
```

flag detritus fluxes in soma for output, set-up remineralisation scheme

Definition at line 108 of file [det.f90](#).

References [det_fluxes\(\)](#), and [det_loss_pic\(\)](#).

Here is the call graph for this function:



5.7 dic Module Reference

Functions for the carbonate system and alkalinity.

Data Types

- type [dicsys](#)

Functions/Subroutines

- subroutine [asfco2](#) (dic, env, box)
Air-to-sea flux of CO₂ and O₂.
- subroutine [co2](#) (dic, box, niter)
Concentrations of protons and DIC species and aragonite and calcite saturation.
- subroutine [dic_potalk](#) (dic, box)
alkalinity as a function of potential alkalinity and DIN
- subroutine [dicalp](#) (dic, box)
parameters for the DIC system and alkalinity (total scale)
- real(dp) elemental function [eos80_rho](#) (tmp, sal)
seawater density in g m⁻³ after [Millero et al. \(1980\)](#)
- elemental real(dp) function [faco2](#) (tk, prat, xco2)
Fugacity of CO₂ in air.
- subroutine [hini_f06](#) (dic, box)
Initialise DIC system after [Follows et al. \(2006\)](#).
- subroutine [hini_m13](#) (dic, box)
Initial guess for [H⁺] after [Munhoven \(2013\)](#).
- subroutine [hsolve_f06](#) (dic, box, niter)
Concentrations of H⁺ from alkalinity and total DIC concentration [H⁺] is calculated after [Follows et al. \(2006\)](#) but including sulphate and fluoride systems.
- subroutine [hsolve_m13](#) (dic, box, niter)
Solve for [H⁺] given alkalinity and total DIC concentration.
- subroutine [indic](#) (dics, lun, env, fluxes)
Initialise DIC module.
- elemental real(dp) function [k0co2](#) (box)
CO₂ solubility
- elemental real(dp) function [lgkspa](#) (sal, tk, tc, prs)
Equilibrium constant for aragonite after [Millero \(1995\)](#).
- elemental real(dp) function [lgkspc](#) (sal, tk, tc, prs)
Equilibrium constant for calcite after [Millero \(1995\)](#).
- elemental real(dp) function [lnk1_sws](#) (sal, tk, lntk, tc, prs, prt, ds, sqs)
 $k1 = [H^+][HCO_3^-]/[CO_2]$: equilibrium constant for CO₂ and HCO₃⁻ in mol kg⁻¹ for SWS
- elemental real(dp) function [lnk1_total](#) (sal, tk, lntk, tc, prs, prt, ds)
 $k1 = [H^+][HCO_3^-]/[CO_2]$: equilibrium constant for CO₂ and HCO₃⁻ in mol kg⁻¹ for the total scale
- elemental real(dp) function [lnk1p_sws](#) (sal, tk, lntk, tc, prs, prt, sqs)
First equilibrium constant of the phosphate system.
- elemental real(dp) function [lnk2_sws](#) (sal, tk, lntk, tc, prs, prt, ds, sqs)
 $k2 = [H^+][CO_3^{2-}]/[HCO_3^-]$: equilibrium constant for HCO₃⁻ and CO₃²⁻ in mol kg⁻¹ for SWS
- elemental real(dp) function [lnk2_total](#) (sal, tk, lntk, tc, prs, prt, ds)
 $k2 = [H^+][CO_3^{2-}]/[HCO_3^-]$: equilibrium constant for HCO₃⁻ and CO₃²⁻ in mol kg⁻¹ for the total scale
- elemental real(dp) function [lnk2p_sws](#) (sal, tk, lntk, tc, prs, prt, sqs)
Second equilibrium constant of the phosphate system.
- elemental real(dp) function [lnk3p_sws](#) (sal, tk, tc, prs, prt, sqs)
Third equilibrium constant of the phosphate system.
- elemental real(dp) function [lnkb_total](#) (sal, tk, lntk, tc, prs, prt, ds, sqs)
Boron system.

- elemental real(dp) function [lnkf_free](#) (tk, tc, prs, prt, i0)
Flouride system (free scale)
- elemental real(dp) function [lnkf_total](#) (tk, tc, prs, prt, sqs)
Flouride system (total scale)
- elemental real(dp) function [lnknh4_sws](#) (sal, tk, tc, prs, prt, sqs)
Ammonium system (SWS)
- elemental real(dp) function [lnks_free](#) (tk, lntk, tc, prs, prt, i0)
Sulfate system.
- elemental real(dp) function [lnkw_sws](#) (sal, tk, lntk, tc, prs, prt, sqs)
Carbonate SYSTEM.
- subroutine [pivel_lm86](#) (dic, env, box)
Piston velocity after [Liss and Merlivat \(1986\)](#).
- subroutine [pivel_w14](#) (dic, env, box)
Piston velocity after [Wanninkhof \(2014\)](#).
- subroutine [pivel_w92](#) (dic, env, box)
Piston velocity after [Wanninkhof \(1992\)](#).
- subroutine [pivel_w99](#) (dic, env, box)
Piston velocity after [Wanninkhof and McGillis \(1999\)](#).

Variables

- real(dp), parameter [rgas](#) = 83.1446261815324_dp
ideal gas constant in dPa m³ K⁻¹ mol⁻¹ (this value is from wikipedia, citing 2018 CODAT, NIST)
- real(dp), parameter [t0ck](#) = 273.15_dp
0 °C in K

5.7.1 Detailed Description

Functions for the carbonate system and alkalinity.

Expressions for the equilibrium constants are from SolveSAPHE ([Munhoven \(2013\)](#)) except for ammonia. Concentrations are converted from mmol m⁻³ to mol kg⁻¹ in subroutine [dicalp](#).

variable	description
at	total alkalinity in eq kg ⁻¹
	at = [HCO ₃ ⁻] + 2 [CO ₃ ²⁻] + [B(OH) ₄ ⁻] + [OH ⁻] + [SiO(OH) ₃ ⁻] + [HS ⁻]
	- ([H ⁺]F + [HSO ₄ ⁻] + [HF])
	H ₂ S and H ₃ PO ₄ systems not included
bt	total boron concentration in mol kg ⁻¹
co2	CO ₂ concentration in mol kg ⁻¹
co3	CO ₃ ²⁻ concentration in mol kg ⁻¹
ct	DIC concentration in mol kg ⁻¹
f	CO ₂ fugacity in Pa
fa	CO ₂ fugacity in air in Pa
hc	proton concentration in mol kg ⁻¹
hco3	HCO ₃ ⁻ concentration in mol kg ⁻¹
i0	ionic strength
k0	solubility of CO ₂ in sea water in mol kg ⁻¹ Pa ⁻¹

variable	description
	$k_0 = [\text{CO}_2]/f\text{CO}_2$
k1	first dissociation constant of the carbonate system in mol kg^{-1} $k_1 = [\text{H}^+][\text{HCO}_3^-]/[\text{CO}_2]$
k1p	first dissociation constant of the phosphate system in mol kg^{-1} $k_{1p} = [\text{H}^+][\text{H}_2\text{PO}_4^-]/[\text{H}_3\text{PO}_4]$
k2	second dissociation constant of the carbonate system in mol kg^{-1} $k_2 = [\text{H}^+][\text{CO}_3^{2-}]/[\text{HCO}_3^-]$
k2p	second dissociation constant of the phosphate system in mol kg^{-1} $k_{2p} = [\text{H}^+][\text{HPO}_4^{2-}]/[\text{H}_2\text{PO}_4^-]$
k3p	third dissociation constant of the phosphate system in mol kg^{-1} $k_{3p} = [\text{H}^+][\text{PO}_4^{3-}]/[\text{HPO}_4^{2-}]$
kb	equilibrium constant of the borate system in mol kg^{-1} $k_b = [\text{H}^+][\text{B}(\text{OH})_4^-]/[\text{B}(\text{OH})_3]$
kf	dissociation constant of hydrogen fluoride in mol kg^{-1} $k_f = [\text{H}^+][\text{F}^-]/[\text{HF}]$
knh4	dissociation constant of ammonium in mol kg^{-1} $k_{nh4} = [\text{H}^+][\text{NH}_3]/[\text{NH}_4^+]$
kso4	equilibrium constant of the sulfate system in mol kg^{-1} $k_{so4} = [\text{H}^+][\text{F}][\text{SO}_4^{2-}]/[\text{HSO}_4^-]$
ksi	equilibrium constant of the silicate system in mol kg^{-1} $k_{si} = [\text{H}^+][\text{SiO}(\text{OH})_3^-]/[\text{Si}(\text{OH})_4]$
kw	equilibrium constant for water in $\text{mol}^2 \text{kg}^{-2}$ $k_w = [\text{H}^+][\text{OH}^-]$
kspa	equilibrium constant for aragonite in mol kg^{-1} $k_{spc} = [\text{Ca}^{2+}][\text{CO}_3^{2-}]$
kspc	equilibrium constant for calcite in mol kg^{-1} $k_{spc} = [\text{Ca}^{2+}][\text{CO}_3^{2-}]$
lnx	$\ln(x)$
prat	atmospheric pressure in Pa
pt	total inorganic phosphorus concentration in mol kg^{-1}
Rgas	ideal gas constant in $\text{dJ K}^{-1} \text{mol}^{-1} = \text{dPa m}^3 \text{K}^{-1} \text{mol}^{-1}$
sal	salinity in psu
schnco2	Schmidt number for CO_2
schno2	Schmidt number for O_2
SO4	total sulfate concentration in mol kg^{-1}
tk	temperature in K
tv	CO_2 air-sea gas transfer velocity (piston velocity) in m s^{-1}
xco2	mole fraction of CO_2 in air

5.7.2 Function/Subroutine Documentation

5.7.2.1 asfco2()

```
subroutine dic::asfco2 (
    class(dicsys), intent(inout) dic,
```



```

type(local), intent(in) env,
type(layer), intent(inout) box ) [private]

```

Air-to-sea flux of CO₂ and O₂.

Oxygen saturation concentration is calculated after [Garcia and Gordon \(1992\)](#), following their recommendation to use the first column of their Table 1.

- See also OCMIP-2

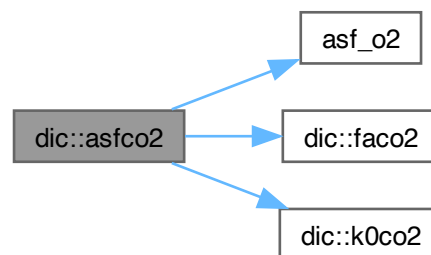
Parameters

in	<i>box%salinity</i>	salinity in psu
in	<i>box%temperature</i>	temperature in °C
in	<i>env%pressure</i>	atmospheric pressure in Pa
in	<i>env%xcO2</i>	atmospheric CO ₂ mole fraction
in	<i>env%wind</i>	wind speed at 10 m above surface in m s ⁻¹
in	<i>dic%co2</i>	concentration of CO ₂ in mol kg ⁻¹
out	<i>dic%facO2</i>	fugacity of CO ₂ in air in Pa
out	<i>dic%k0co2</i>	solubility of CO ₂ in mol kg ⁻¹ Pa ⁻¹
out	<i>dic%asfco2</i>	air-sea flux of CO ₂ in mmol m ⁻² s ⁻¹
out	<i>dic%asfo2</i>	air-sea flux of O ₂ in mmol m ⁻² s ⁻¹

Definition at line 192 of file [dic.f90](#).

References [asf_o2\(\)](#), [facO2\(\)](#), and [k0co2\(\)](#).

Here is the call graph for this function:



5.7.2.2 co2()

```

subroutine dic::co2 (
    class(dicsys), intent(inout) dic,
    type(layer), intent(inout) box,
    integer, intent(in) niter ) [private]

```

Concentrations of protons and DIC species and aragonite and calcite saturation.

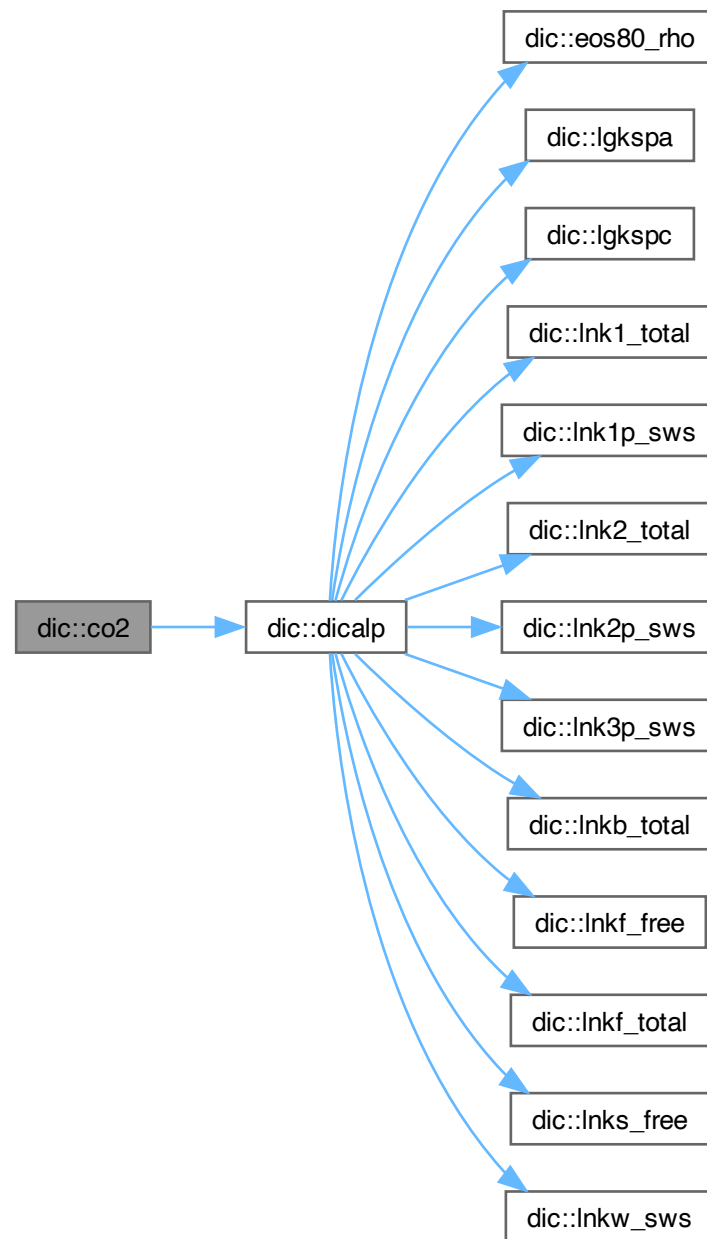
Parameters

in	<i>box%alkalinity</i>	total alkalinity in $\mu\text{mol L}^{-1}$ (or boxDIN and dicpotalk)
in	<i>box%salinity</i>	salinity in psu
in	<i>box%temperature</i>	temperature in $^{\circ}\text{C}$
in	<i>box%DIC</i>	DIC concentration in mmol m^{-3}
in	<i>box%DIP</i>	phosphate concentration in mmol m^{-3}
in	<i>box%silc</i>	silicate concentration in mmol m^{-3}
in, out	<i>box%hc</i>	proton concentration in mol kg^{-1}
out	<i>box%tk</i>	temperature in K
out	<i>box%CO2</i>	CO_2 concentration in mol kg^{-1}
out	<i>box%HCO3</i>	HCO_3^- concentration in mol kg^{-1}
out	<i>box%CO3</i>	CO_3^{2-} concentration in mol kg^{-1}
out	<i>box%rho</i>	density in g m^{-3}
out	<i>box%omega</i>	aragonite saturation, Millero (1995) , Eq. (81)
out	<i>box%omega_calcite</i>	calcite saturation, Millero (1995) , Eq. (80)

Definition at line 670 of file [dic.f90](#).

References [dicalp\(\)](#).

Here is the call graph for this function:



5.7.2.3 dic_potalk()

```

subroutine dic::dic_potalk (
    class(dicsys), intent(in) dic,
    type(layer), intent(inout) box ) [private]

```

alkalinity as a function of potential alkalinity and DIN

Definition at line 723 of file [dic.f90](#).

Referenced by [indic\(\)](#).

Here is the caller graph for this function:



5.7.2.4 dicalp()

```

subroutine dic::dicalp (
    class(dicsys), intent(inout) dic,
    type(layer), intent(inout) box ) [private]
  
```

parameters for the DIC system and alkalinity (total scale)

Parameters

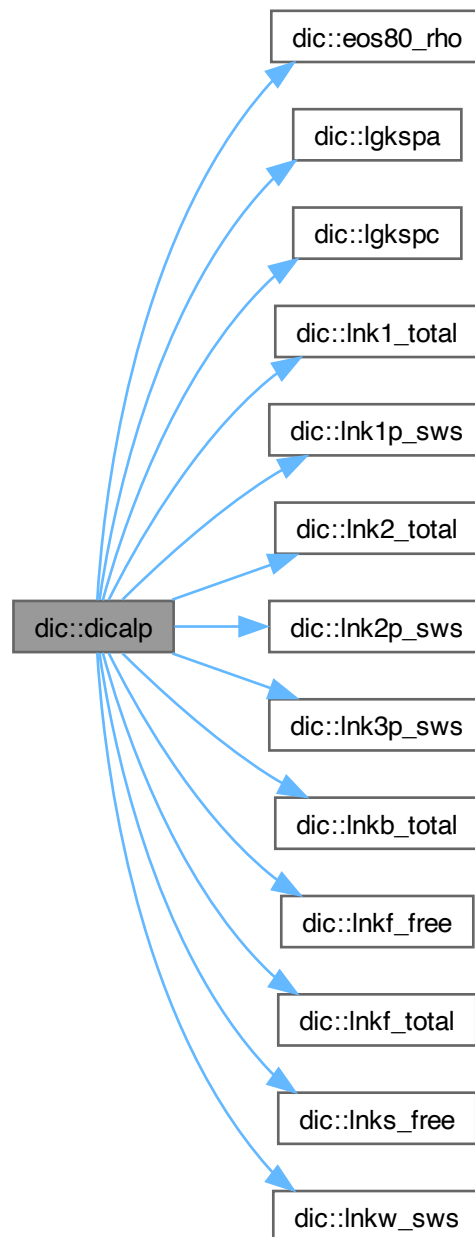
in	<i>box%tk</i>	temperature in K
in	<i>box%salinity</i>	salinity in psu
in	<i>box%alkalinity</i>	alkalinity in mmol m^{-3}
in	<i>box%depth</i>	depth in m
in	<i>box%DIC</i>	DIC in mmol m^{-3}
out	<i>box%rho</i>	density in g m^{-3}
out	<i>box%at</i>	alkalinity in mol kg^{-1}
out	<i>box%ct</i>	DIC in mol kg^{-1}

Definition at line 596 of file [dic.f90](#).

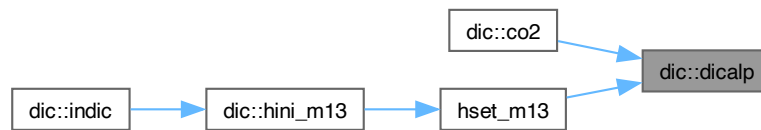
References [eos80_rho\(\)](#), [lgkspa\(\)](#), [lgkspc\(\)](#), [lnk1_total\(\)](#), [lnk1p_sws\(\)](#), [lnk2_total\(\)](#), [lnk2p_sws\(\)](#), [lnk3p_sws\(\)](#), [lnkb_total\(\)](#), [lnkf_free\(\)](#), [lnkf_total\(\)](#), [lnks_free\(\)](#), [lnkw_sws\(\)](#), [rgas](#), and [t0ck](#).

Referenced by [co2\(\)](#), and [hset_m13\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.2.5 eos80_rho()

```

real(dp) elemental function dic::eos80_rho (
    real(dp), intent(in) tmp,
    real(dp), intent(in) sal ) [private]
  
```

seawater density in g m^{-3} after [Millero et al. \(1980\)](#)

Parameters

in	<i>tmp</i>	temperature in °C
in	<i>sal</i>	salinity in psu

Definition at line 734 of file [dic.f90](#).

Referenced by [dicalp\(\)](#).

Here is the caller graph for this function:



5.7.2.6 faco2()

```

elemental real(dp) function dic::faco2 (
    real(dp), intent(in) tk,
    real(dp), intent(in) prat,
    real(dp), intent(in) xco2 ) [private]
  
```

Fugacity of CO₂ in air.

Equations from SOP 24 in [Dickson and Goyet \(1997\)](#). See also OCMIP-2

Parameters

in	<i>tk</i>	temperature in K
in	<i>prat</i>	atmospheric pressure in Pa
in	<i>xco2</i>	atmospheric CO ₂ mole fraction

Returns

fugacity of CO₂ in air in Pa

Parameters

in	<i>tk</i>	temperature in K
in	<i>prat</i>	atmospheric pressure in Pa
in	<i>xco2</i>	atmospheric CO ₂ mole fraction

Definition at line 337 of file [dic.f90](#).

References [rgas](#).

Referenced by [asfco2\(\)](#).

Here is the caller graph for this function:



5.7.2.7 hini_f06()

```

subroutine dic::hini_f06 (
    class(dicsys), intent(inout) dic,
    type(layer), dimension(0:), intent(inout) box ) [private]
  
```

Initialise DIC system after [Follows et al. \(2006\)](#).

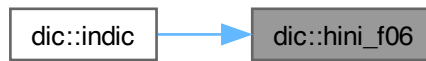
Parameters

out	<i>box%hc</i>	initial estimate for [H ⁺] in mol kg ⁻¹
-----	---------------	--

Definition at line 688 of file [dic.f90](#).

Referenced by [indic\(\)](#).

Here is the caller graph for this function:



5.7.2.8 hini_m13()

```
subroutine dic::hini_m13 (  
    class(dicsys), intent(inout) dic,  
    type(layer), dimension(0:), intent(inout) box ) [private]
```

Initial guess for [H+] after [Munhoven \(2013\)](#).

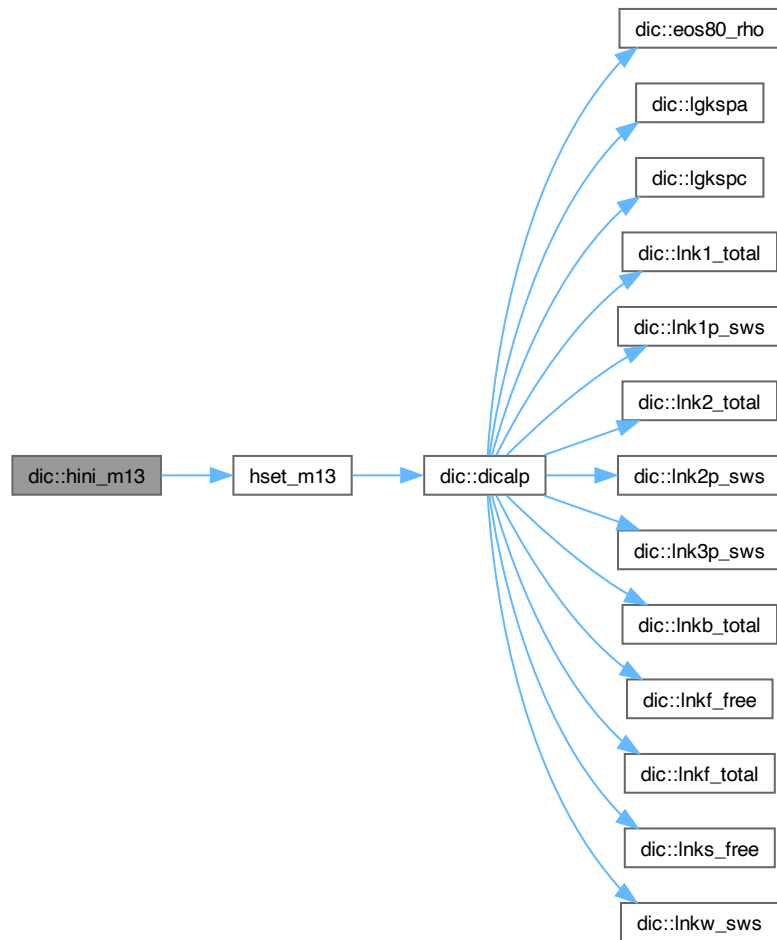
This is an adaptation of subroutine AHINI_FOR_AT from [Munhoven \(2013\)](#). Total alkalinity (boxalkalinity) is used as an approximation for carbonate alkalinity.

Definition at line [748](#) of file [dic.f90](#).

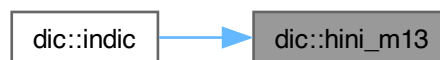
References [hset_m13\(\)](#).

Referenced by [indic\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.2.9 hsolve_f06()

```

subroutine dic::hsolve_f06 (
    class(dicsys), intent(inout) dic,

```

```

type(layer), intent(inout) box,
integer, intent(in) niter ) [private]

```

Concentrations of H^+ from alkalinity and total DIC concentration $[H^+]$ is calculated after [Follows et al. \(2006\)](#) but including sulphate and fluoride systems.

Definition at line 701 of file [dic.f90](#).

Referenced by [indic\(\)](#).

Here is the caller graph for this function:



5.7.2.10 hsolve_m13()

```

subroutine dic::hsolve_m13 (
    class(dicsys), intent(inout) dic,
    type(layer), intent(inout) box,
    integer, intent(in) niter ) [private]

```

Solve for $[H^+]$ given alkalinity and total DIC concentration.

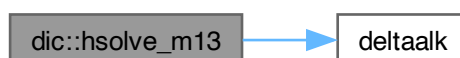
This is an adaptation of function SOLVE_AT_GENERAL_SEC (without ammonium and H_2S) from [Munhoven \(2013\)](#) for the total pH scale. Munhoven's routine has more iterations but here the result from the previous call is used as the initial guess. Also, iterations are done only if the error in the alkalinity equation (DeltaAlk) is greater than $1e-9$ (relative error $> 1e-6$).

Definition at line 798 of file [dic.f90](#).

References [deltaalk\(\)](#).

Referenced by [indic\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.7.2.11 indic()

```

subroutine dic::indic (
    class(dicsys), intent(inout) dics,
    integer, intent(in) lun,
    type(local), intent(inout) env,
    real(dp), dimension(:,0:), intent(in), optional, target fluxes )
  
```

Initialise DIC module.

pvfun	piston velocity function
LM86	Liss and Merlivat (1986) (default)
W92	Wanninkhof (1992)
WG99	Wanninkhof and McGillis (1999)
W14	Wanninkhof (2014)

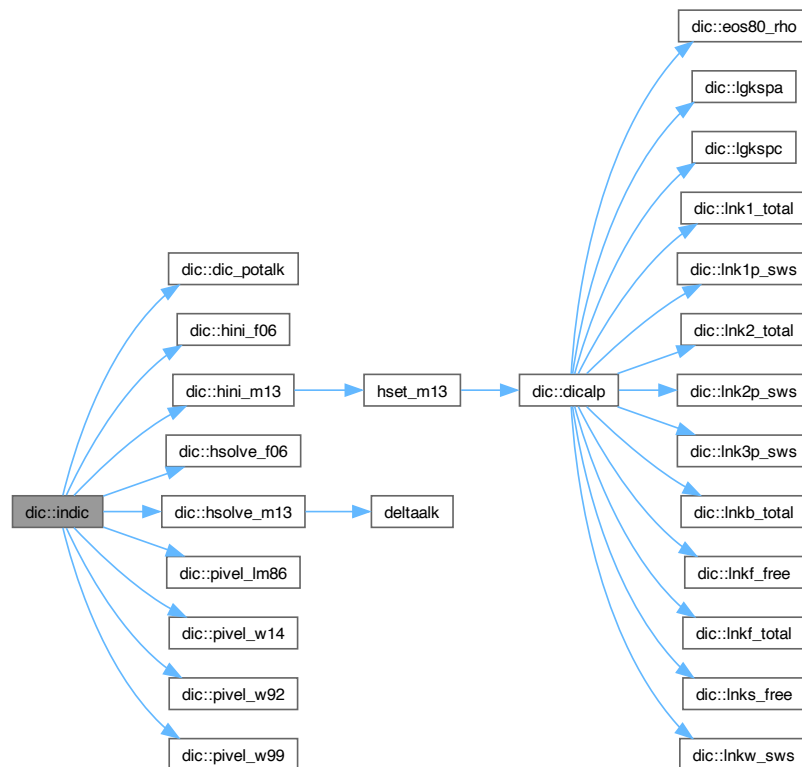
Parameters

in	<i>dics%totalk</i>	total alkalinity in mmol m ⁻³
out	<i>dics%rrnsi</i>	N:Si Redfield ratio
out	<i>dics%rrnp</i>	N:P Redfield ratio
out	<i>dics%hc</i>	initial guess for H ⁺ concentration
in	<i>lun</i>	of file with namelist dic

Definition at line 111 of file [dic.f90](#).

References [dic_potalk\(\)](#), [hini_f06\(\)](#), [hini_m13\(\)](#), [hsolve_f06\(\)](#), [hsolve_m13\(\)](#), [pivel_lm86\(\)](#), [pivel_w14\(\)](#), [pivel_w92\(\)](#), and [pivel_w99\(\)](#).

Here is the call graph for this function:



5.7.2.12 k0co2()

```

elemental real(dp) function dic::k0co2 (
    type(layer), intent(in) box ) [private]

```

CO₂ solubility

Equation (7.1.3) from Chapter 5 in *Dickson and Goyet* (1997).

Parameters

in	<i>box%salinity</i>	salinity in psu
in	<i>box%tk</i>	temperature in K

Returns

CO₂ solubility in mol kg⁻¹ Pa⁻¹

Definition at line 319 of file [dic.f90](#).

Referenced by [asfco2\(\)](#).

Here is the caller graph for this function:



5.7.2.13 lgkspa()

```

elemental real(dp) function dic::lgkspa (
    real(dp), intent(in) sal,
    real(dp), intent(in) tk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs ) [private]
  
```

Equilibrium constant for aragonite after [Millero \(1995\)](#).

Parameters

in	<i>sal</i>	salinity in psu
in	<i>tk</i>	temperature in K
in	<i>tc</i>	temperature in °C
in	<i>prs</i>	pressure in bar

Returns

common logarithm of the equilibrium constant for aragonite

Definition at line 558 of file [dic.f90](#).

References [rgas](#).

Referenced by [dicalp\(\)](#).

Here is the caller graph for this function:



5.7.2.14 lgkspc()

```

elemental real(dp) function dic::lgkspc (
    real(dp), intent(in) sal,
    real(dp), intent(in) tk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs ) [private]

```

Equilibrium constant for calcite after [Millero \(1995\)](#).

Parameters

in	<i>sal</i>	salinity in psu
in	<i>tk</i>	temperature in K
in	<i>tc</i>	temperature in °C
in	<i>prs</i>	pressure in bar

Returns

common logarithm of the equilibrium constant for calcite

Definition at line 576 of file [dic.f90](#).

References [rgas](#).

Referenced by [dicalp\(\)](#).

Here is the caller graph for this function:



5.7.2.15 lnk1_sws()

```

elemental real(dp) function dic::lnk1_sws (
    real(dp), intent(in) sal,
    real(dp), intent(in) tk,
    real(dp), intent(in) lntk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs,
    real(dp), intent(in) prt,
    real(dp), intent(in) ds,
    real(dp), intent(in) sqs ) [private]

```

$k_1 = [\text{H}^+][\text{HCO}_3^-]/[\text{CO}_2]$: equilibrium constant for CO_2 and HCO_3^- in mol kg^{-1} for SWS

Parameters

in	prs	pressure in bar
----	-----	-----------------

Definition at line 370 of file [dic.f90](#).

5.7.2.16 `lnk1_total()`

```

elemental real(dp) function dic::lnk1_total (
    real(dp), intent(in) sal,
    real(dp), intent(in) tk,
    real(dp), intent(in) lntk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs,
    real(dp), intent(in) prt,
    real(dp), intent(in) ds ) [private]

```

$k_1 = [\text{H}^+][\text{HCO}_3^-]/[\text{CO}_2]$: equilibrium constant for CO_2 and HCO_3^- in mol kg^{-1} for the total scale

Parameters

in	prs	pressure in bar
----	-----	-----------------

Definition at line 398 of file [dic.f90](#).

Referenced by [dicalp\(\)](#).

Here is the caller graph for this function:

5.7.2.17 `lnk1p_sws()`

```

elemental real(dp) function dic::lnk1p_sws (
    real(dp), intent(in) sal,
    real(dp), intent(in) tk,
    real(dp), intent(in) lntk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs,
    real(dp), intent(in) prt,
    real(dp), intent(in) sqs ) [private]

```

First equilibrium constant of the phosphate system.

Definition at line 511 of file [dic.f90](#).

Referenced by [dicalp\(\)](#).

Here is the caller graph for this function:



5.7.2.18 lnk2_sws()

```

elemental real(dp) function dic::lnk2_sws (
    real(dp), intent(in) sal,
    real(dp), intent(in) tk,
    real(dp), intent(in) lntk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs,
    real(dp), intent(in) prt,
    real(dp), intent(in) ds,
    real(dp), intent(in) sqs ) [private]

```

$k_2 = [\text{H}^+][\text{CO}_3^{2-}]/[\text{HCO}_3^-]$: equilibrium constant for HCO_3^- and CO_3^{2-} in mol kg^{-1} for SWS

Parameters

in	prs	pressure in bar
----	-----	-----------------

Definition at line 384 of file [dic.f90](#).

5.7.2.19 lnk2_total()

```

elemental real(dp) function dic::lnk2_total (
    real(dp), intent(in) sal,
    real(dp), intent(in) tk,
    real(dp), intent(in) lntk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs,
    real(dp), intent(in) prt,
    real(dp), intent(in) ds ) [private]

```

$k_2 = [\text{H}^+][\text{CO}_3^{2-}]/[\text{HCO}_3^-]$: equilibrium constant for HCO_3^- and CO_3^{2-} in mol kg^{-1} for the total scale

Parameters

in	prs	pressure in bar
----	-----	-----------------

Definition at line 412 of file [dic.f90](#).

Referenced by [dicalp\(\)](#).

Here is the caller graph for this function:



5.7.2.20 lnk2p_sws()

```

elemental real(dp) function dic::lnk2p_sws (
    real(dp), intent(in) sal,
    real(dp), intent(in) tk,
    real(dp), intent(in) lntk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs,
    real(dp), intent(in) prt,
    real(dp), intent(in) sqs ) [private]

```

Second equilibrium constant of the phosphate system.

Parameters

in	prs	pressure in bar
----	-----	-----------------

Definition at line 525 of file [dic.f90](#).

Referenced by [dicalp\(\)](#).

Here is the caller graph for this function:



5.7.2.21 lnk3p_sws()

```

elemental real(dp) function dic::lnk3p_sws (
    real(dp), intent(in) sal,

```

```

real(dp), intent(in) tk,
real(dp), intent(in) tc,
real(dp), intent(in) prs,
real(dp), intent(in) prt,
real(dp), intent(in) sqs ) [private]

```

Third equilibrium constant of the phosphate system.

Parameters

in	<i>prs</i>	pressure in bar
in	<i>prt</i>	(pressure in bar)/(R*T)

Definition at line 540 of file [dic.f90](#).

Referenced by [dicalp\(\)](#).

Here is the caller graph for this function:



5.7.2.22 lnkb_total()

```

elemental real(dp) function dic::lnkb_total (
    real(dp), intent(in) sal,
    real(dp), intent(in) tk,
    real(dp), intent(in) lntk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs,
    real(dp), intent(in) prt,
    real(dp), intent(in) ds,
    real(dp), intent(in) sqs ) [private]

```

Boron system.

$kb = [H^+][B(OH)_4^-]/[B(OH)_3]$: equilibrium constant of boric acid system
in mol/kg

Parameters

in	<i>prs</i>	pressure in bar
----	------------	-----------------

Definition at line 447 of file [dic.f90](#).

Referenced by [dicalp\(\)](#).

Here is the caller graph for this function:



5.7.2.23 lnkf_free()

```

elemental real(dp) function dic::lnkf_free (
    real(dp), intent(in) tk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs,
    real(dp), intent(in) prt,
    real(dp), intent(in) i0 ) [private]
  
```

Flouride system (free scale)

$khf(f) = [H+][F-]/[HF]$: equilibrium constant of fluoric acid system in mol/kg

Parameters

in	prs	pressure in bar
----	-----	-----------------

Definition at line 480 of file [dic.f90](#).

Referenced by [dicalp\(\)](#).

Here is the caller graph for this function:



5.7.2.24 lnkf_total()

```

elemental real(dp) function dic::lnkf_total (
    real(dp), intent(in) tk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs,
    real(dp), intent(in) prt,
    real(dp), intent(in) sqs ) [private]
  
```

Flouride system (total scale)

$khf = [H+][F-]/[HF]$: equilibrium constant of fluoric acid system in mol/kg

Parameters

<i>in</i>	<i>prs</i>	pressure in bar
-----------	------------	-----------------

Definition at line 495 of file [dic.f90](#).

Referenced by [dicalp\(\)](#).

Here is the caller graph for this function:

**5.7.2.25 lnkh4_sws()**

```

elemental real(dp) function dic::lnkh4_sws (
    real(dp), intent(in) sal,
    real(dp), intent(in) tk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs,
    real(dp), intent(in) prt,
    real(dp), intent(in) sqs ) [private]

```

Ammonium system (SWS)

$\text{khf}(f) = [\text{H}^+][\text{F}^-]/[\text{HF}]$: equilibrium constant of fluoric acid system in mol/kg

Parameters

<i>in</i>	<i>prs</i>	pressure in bar
-----------	------------	-----------------

Definition at line 464 of file [dic.f90](#).

5.7.2.26 lnks_free()

```

elemental real(dp) function dic::lnks_free (
    real(dp), intent(in) tk,
    real(dp), intent(in) lntk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs,
    real(dp), intent(in) prt,
    real(dp), intent(in) i0 ) [private]

```

Sulfate system.

$\text{kSO4} = [\text{H}^+][\text{SO}_4^{2-}]/[\text{HSO}_4^-]$: equilibrium constant of sulfate system in mol kg^{-1}

Parameters

in	<i>prs</i>	pressure in bar
in	<i>i0</i>	ionic strength

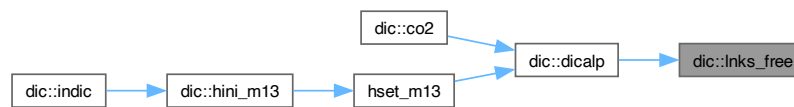
Returns

natural logarithm of k_{SO4}

Definition at line 429 of file [dic.f90](#).

Referenced by [dicalp\(\)](#).

Here is the caller graph for this function:



5.7.2.27 lnkw_sws()

```

elemental real(dp) function dic::lnkw_sws (
    real(dp), intent(in) sal,
    real(dp), intent(in) tk,
    real(dp), intent(in) lntk,
    real(dp), intent(in) tc,
    real(dp), intent(in) prs,
    real(dp), intent(in) prt,
    real(dp), intent(in) sqs ) [private]

```

Carbonate SYSTEM.

$k_w = [H^+][OH^-]$: equilibrium constant for water in $\text{mol}^2 \text{kg}^{-2}$

Parameters

in	<i>prs</i>	pressure in bar
----	------------	-----------------

Definition at line 356 of file [dic.f90](#).

Referenced by [dicalp\(\)](#).

Here is the caller graph for this function:



5.7.2.28 pivel_lm86()

```

subroutine dic::pivel_lm86 (
    class(dicsys), intent(inout) dic,
    type(local), intent(in) env,
    type(layer), intent(in) box ) [private]

```

Piston velocity after [Liss and Merlivat \(1986\)](#).

Parameters

in	<i>env%wind</i>	wind speed at 10 m in m s^{-1}
out	<i>dic%tvco2</i>	transfer (piston) velocity for CO_2 in m s^{-1}
in	<i>dic%tvo2</i>	transfer (piston) velocity for O_2 in m s^{-1}

Definition at line 224 of file [dic.f90](#).

Referenced by [indlc\(\)](#).

Here is the caller graph for this function:



5.7.2.29 pivel_w14()

```

subroutine dic::pivel_w14 (
    class(dicsys), intent(inout) dic,
    type(local), intent(in) env,
    type(layer), intent(in) box ) [private]

```

Piston velocity after [Wanninkhof \(2014\)](#).

Parameters

in	<i>box%temperature</i>	sea-surface temperature in °C
in	<i>env%wind</i>	wind speed at 10 m in m s ⁻¹
out	<i>dic%schnco2</i>	Schmidt No. for CO ₂
out	<i>dic%schno2</i>	Schmidt No. for O ₂
out	<i>dic%tvco2</i>	transfer (piston) velocity for CO ₂ in m s ⁻¹
out	<i>dic%tvo2</i>	transfer (piston) velocity for O ₂ in m s ⁻¹

Definition at line 297 of file [dic.f90](#).

Referenced by [indic\(\)](#).

Here is the caller graph for this function:



5.7.2.30 pivel_w92()

```

subroutine dic::pivel_w92 (
    class(dicsys), intent(inout) dic,
    type(local), intent(in) env,
    type(layer), intent(in) box ) [private]
  
```

Piston velocity after [Wanninkhof \(1992\)](#).

Parameters

in	<i>box%temperature</i>	sea-surface temperature in °C
in	<i>env%wind</i>	wind speed at 10 m in m s ⁻¹
out	<i>dic%schnco2</i>	Schmidt No. for CO ₂
out	<i>dic%schno2</i>	Schmidt No. for O ₂
out	<i>dic%tvco2</i>	transfer (piston) velocity for CO ₂ in m s ⁻¹
out	<i>dic%tvo2</i>	transfer (piston) velocity for O ₂ in m s ⁻¹

Definition at line 247 of file [dic.f90](#).

Referenced by [indic\(\)](#).

Here is the caller graph for this function:



5.7.2.31 pivel_w99()

```

subroutine dic::pivel_w99 (
    class(dicsys), intent(inout) dic,
    type(local), intent(in) env,
    type(layer), intent(in) box ) [private]
  
```

Piston velocity after [Wanninkhof and McGillis \(1999\)](#).

Parameters

in	<i>box%temperature</i>	sea-surface temperature in °C
in	<i>env%wind</i>	wind speed at 10 m in m s ⁻¹
out	<i>dic%schnco2</i>	Schmidt No. for CO ₂
out	<i>dic%schno2</i>	Schmidt No. for O ₂
out	<i>dic%tvco2</i>	transfer (piston) velocity for CO ₂ in m s ⁻¹
out	<i>dic%tvo2</i>	transfer (piston) velocity for O ₂ in m s ⁻¹

Definition at line 272 of file [dic.f90](#).

Referenced by [indic\(\)](#).

Here is the caller graph for this function:



5.7.3 Variable Documentation

5.7.3.1 rgas

```

real(dp), parameter dic::rgas = 83.1446261815324_dp [private]
  
```


ideal gas constant in dPa m³ K⁻¹ mol⁻¹ (this value is from wikipedia, citing 2018 CODAT, NIST)

Definition at line 67 of file [dic.f90](#).

Referenced by [dicalp\(\)](#), [faco2\(\)](#), [lgkspa\(\)](#), and [lgkspc\(\)](#).

5.7.3.2 t0ck

```
real(dp), parameter dic::t0ck =273.15_dp [private]
```

0 °C in K

Definition at line 66 of file [dic.f90](#).

Referenced by [asf_o2\(\)](#), and [dicalp\(\)](#).

5.8 dcode Module Reference

VODE_F90 interface for oppla.

Data Types

- type [ode](#)

Functions/Subroutines

- subroutine [read_ode](#) (parode, lun, nconstr)
Read DVODE parameters.
- subroutine [set_atol](#) (parode, stv)
Set absolute tolerances for each state individually.

5.8.1 Detailed Description

VODE_F90 interface for oppla.

5.8.2 Function/Subroutine Documentation

5.8.2.1 read_ode()

```
subroutine dcode::read_ode (
    class(ode), intent(inout) parode,
    integer, intent(in) lun,
    integer, intent(in) nconstr )
```

Read DVODE parameters.

Parameters

in	<i>lun</i>	logical unit number for namelist vode
in	<i>nconstr</i>	number of constrained states

Definition at line 30 of file [dcode.f90](#).

References [stdunt::stderr](#).

5.8.2.2 set_atol()

```
subroutine dcode::set_atol (
    class(ode), intent(inout) parode,
    real(dp), dimension(*), intent(in) stv ) [private]
```

Set absolute tolerances for each state individually.

Absolute tolerances atol and mxatol are assigned individually if provided as negative values in namelist vode.

Definition at line 113 of file [dcode.f90](#).

5.9 et Module Reference

Ecological types.

Data Types

- interface [aggregation](#)
Calculate aggregation fluxes.
- interface [attenuate](#)
Light attenuation.
- interface [boxalk](#)
- interface [boxbc](#)
- interface [decl](#)
daylength function
- interface [fluxes](#)
Calculate fluxes in the source matrix soma.
- interface [fpar](#)
Fractions of surface PAR arriving at the box bottoms.
- type [fungroup](#)
- interface [init](#)
Read namelists.
- type [layer](#)
Conditions and fluxes local to each box (layer) [More...](#)
- interface [light](#)
Calculate or read surface irradiance.
- type [local](#)
- interface [preset](#)
Define ratios, set indices, pointers.
- type [state](#)
- interface [sunday](#)
- type [timing](#)

Functions/Subroutines

- real(dp) function [ft_eppley](#) (grp, box)
Temperature function after Eppley (1972)
- real(dp) function [ft_houlton](#) (grp, box)
Temperature function after Houlton et al. (2008)
- real(dp) function [ft_me](#) (grp, box)
Temperature function for Mytilus edulis.
- real(dp) function [ft_oppla](#) (grp, box)
Temperature function based on Moisan et al. (2002)
- subroutine [ft_select](#) (grp, ft, caller)
Select and associate temperature-dependence function.

Variables

- character(len= *), dimension(*), parameter [constituent_units](#) = ('mmol m-3 ', 'mmol m-3 ', 'mmol m-3 ', 'mmol m-3 ', 'mgChl m-3'/)
- character(len= *), dimension(*), parameter [constituents](#) = ('C ', 'N ', 'P ', 'PIC', 'Chl'/)
- real(dp), parameter [pi](#) = 3.14159265358979323_dp
Common components for all functional groups.
- real(dp), parameter [rad](#) = pi/180._dp

5.9.1 Detailed Description

Ecological types.

This module defines four types: group, state, local, layer, and timing. The types group and state form the core of representation of the plankton community. The state type has only one component, var, which is an allocatable array of pointers of class group. The group type is extended in the modules for the functional types, cfo, cmo, bac, etc. Each element of the var array represents one functional group of the model plankton ecosystem.

This module also provides a suite of temperature functions and the subroutine [ft_select](#), which are used by the functional-group modules to select temperature functions for individual processes.

5.9.2 Data Type Documentation

5.9.2.1 type `et::layer`

Conditions and fluxes local to each box (layer)

- `stv` is a pointer to the state variables in the current box, i.e., [stv\(nb,:\)](#) in module `plankton` and `stvdatt(nb,:)` in subroutine `invar` (module `onf`)
- `soma` is a pointer to the source matrix of fluxes among state variables within the current box (layer); the first index denotes the source, the second index the target of the flux; diagonal elements store rates of change not affecting other state variables; the first row (first index = 0) is for aggregation

Definition at line [146](#) of file [et.f90](#).

Class Members

real(dp), pointer	alkalinity	
real(dp), dimension(:), allocatable	bbc	bottom boundary conditions
real(dp)	botperm =1._dp	bottom permeability (for sinking etc.)
real(dp), pointer	bpar	irradiance (PAR) at bottom of box
real(dp)	co2 =0._dp	CO2 concentration in mol kg ⁻¹ .
real(dp)	co3	CO3 concentration in mol kg ⁻¹ .
real(dp)	delvel	velocity difference between top and bottom of box
real(dp)	depth	mid-depth of current box in m
real(dp), pointer	dic	
real(dp), pointer	din	
real(dp), pointer	dip	
real(dp)	dpar	depth average of daytime irradiance
real(dp)	dz	distance between current and above boxes mid-depths
integer	fish =0	factor for fish mortality (0 or 1)
real(dp), dimension(:), pointer	flux	vector of vertical fluxes toward the layer beneath
real(dp)	hc	proton concentration in mol kg ⁻¹
real(dp)	hco3	HCO3 concentration in mol kg ⁻¹ .
real(dp)	height =0D0	height of current box in m
real(dp)	ipar	instantaneous depth-averaged irradiance
real(dp)	lch	(light-attenuation coefficient)*(box height)
integer	nb	number of current box (layer); nb = 1 is the surface box
real(dp), pointer	o2	
real(dp)	omega	aragonite saturation
real(dp)	omega_calcite	calcite saturation
real(dp), dimension(:), pointer	ratios	ratios in this layer
real(dp)	rdl	ratio of lower box height to total height of two boxes
real(dp), pointer	rdoc	
real(dp), pointer	rdon	
real(dp)	rho	density in g m ⁻³
real(dp), pointer	salinity	
real(dp), pointer	silc	total Si concentration in mol kg ⁻¹
real(dp), dimension(:, :), pointer	soma	fluxes among states within the current box
real(dp), dimension(:), pointer	stv	state variables in the current box
real(dp), pointer	temperature	
real(dp)	tk	temperature in K
real(dp), pointer	tpar	irradiance (PAR) at top of box
real(dp), pointer	vdc	vertical diffusivity in m ² s ⁻¹
real(dp), pointer	velocity	vertical velocity in m s ⁻¹ (positive downward)

5.9.2.2 type et::local

Definition at line 63 of file [et.f90](#).

Class Members

real(dp), dimension(:), allocatable	agg	
procedure(agggregation), pointer	aggregate	aggregation fluxes
procedure(boxalk), pointer	alkalinity	alkalinity
real(dp), dimension(:, :), allocatable	all_ratios	all ratios in all boxes
procedure(attenuate), pointer	attenuation	light attenuation
real(dp), dimension(:), allocatable	bbc	bottom-boundary conditions for layers
procedure(boxbc), pointer	boxbc	box-boundary concentrations/conditions
logical	calcpa = .TRUE.	should PAR be calculated from clear-sky radiation in sunday_brock81?
real(dp), pointer	daylen	day length fraction
character(len=256)	daylenfun = "	
real(dp), pointer	daypar	average day-time surface PAR
real(dp), dimension(:), allocatable	depth	
real(dp), dimension(:, :), pointer	depth_edges	depths of layer edges
character(len=256)	dicy = "	diurnal light cycle
real(dp), dimension(:), allocatable	dydz	
real(dp), dimension(:), allocatable	falk	alkalinity factors for DIN, PIC, etc.
real(dp), dimension(:, :), allocatable	fluxes	flux vectors
logical, dimension(:, :), allocatable	fmsk	fluxes-matrix mask
real(dp), dimension(:), allocatable	fpar	fraction of sPAR arriving at the top of each layer
real(dp), dimension(:), allocatable	fsfalk	alkalinity factors for surface fluxes
real(dp), dimension(:, :), allocatable	heights	layer heights
integer, dimension(:), allocatable	iagg	
integer	ialk = 0	
real(dp), pointer	ice	ice cover fraction
integer, dimension(:), allocatable	ichl	indices of Chl tracers
integer	idagg = 0	index of dretritus group representing aggregates
integer	idic = 0	
integer	idin = 0	
integer	idip = 0	
integer, dimension(:), allocatable	ifalk	indices of tracers influencing alkalinity
integer	io2 = 0	
integer, dimension(:), allocatable	iphy	
integer, dimension(:), allocatable	ipic	indices of PIC tracers
integer, dimension(:), allocatable	ipoc	indices of POC tracers
integer, dimension(:), allocatable	iqOr	ratio-indices of subsistence quotas
integer, dimension(:), allocatable	irc	indices of C tracers (reference tracers for ratios)
integer	irdoc = 0	
integer	irdon = 0	
integer, dimension(:), allocatable	irt	tracer-indices of quota tracers
integer, dimension(:), allocatable	isfalk	indices of surface fluxes influencing alkalinity
integer	isi = 0	
real(dp), dimension(:, :), allocatable	kij	coagulation kernel matrix
real(dp)	lachl = 16D0	
real(dp), pointer	lacw	light-attenuation coefficient of water in m ⁻¹
real(dp)	lapon = 16D0	light attenuation coefficient of chl, pon

Class Members

integer	nagg	No. of types involved in aggregation.
integer	nbac	
integer	nbox	No. of boxes (layers)
integer	ncon	No. of different material constituents.
integer	ndet	
integer	ndm	No. of dissolved-matter variables except labile DOM.
integer	ndom	No. of DOM groups.
integer	nft	No. of functional types.
integer	nphy	
integer	nq0	No. of subsistence quotas.
integer	nrs	No. of quotas.
integer	nzoo	No. of bacteria, detritus, phytoplankton, zooplankton groups.
real(dp), dimension(:), allocatable	oc	DOC or POC for all groups.
real(dp), dimension(:), allocatable	par	surface PAR for each layer
procedure(fpar), pointer	parfrac	fractions of surface PAR arriving at the box bottoms
procedure(light), pointer	parfun	calculate or read surface irradiance
real(dp), pointer	pressure	atmospheric surface pressure in Pa
real(dp), dimension(:), allocatable	q0	subsistence quotas
logical, dimension(:,,:), allocatable	qmsk	quota mask: .TRUE. if the quota is variable (a dynamic ratio)
real(dp), dimension(:,,:), allocatable	quotas	biomass (C)-normalised quotas for all groups, dimensions are (nq,nft): <code>quotas(1,:) = 1</code> , <code>quotas(2,:) = QN</code> , <code>quotas(3,:) = QP</code> , etc.
real(dp), dimension(:), allocatable	ratios	ratios of all groups
real(dp)	rq = 1.2D0	respiratory quotient in mol O ₂ (mol C) ⁻¹
real(dp)	salinity	constant salinity if no profile data available
real(dp), pointer	sfdic	surface flux of DIC in mmol m ⁻² s ⁻¹
real(dp), pointer	sfo2	surface flux of O ₂ in mmol m ⁻² s ⁻¹
logical, dimension(:,,:), allocatable	smsk	source-matrix mask
real(dp), dimension(:,,:), allocatable	soma	source matrices
real(dp), pointer	spar	instantaneous surface PAR
real(dp), dimension(:), allocatable	sticky	
real(dp)	temperature	constant temperature if no profile data available
real(dp)	vdc	constant vertical mixing if no profile data available
real(dp)	velocity	constant vertical velocity if no profile data available
real(dp), dimension(:), allocatable	vvs	relative vertical velocities (sinking, vertical migration, etc.)
real(dp), dimension(:), allocatable	vvstv	absolute vertical velocities, including up- or downwelling
real(dp), pointer	wind	wind speed in m s ⁻¹
real(dp), pointer	xco2	mole fraction of CO ₂ in air
logical	xsfolk	do any surface fluxes affect alkalinity?

5.9.2.3 type et::state

Definition at line 60 of file [et.f90](#).

Class Members

class(fungroup), pointer	var	pointer to functional group of type bacteria, phycmo, zoocfo, etc.
--	-----	--

5.9.2.4 type et::timing

Definition at line 179 of file [et.f90](#).

Class Members

real(dp)	cdl	$\text{COS}(d1) * \text{COS}(latrad)$
integer(int64)	current	current model time in s during integration in subroutine plankton:plankton_ode
real(dp)	d1	declination
real(dp)	daylen =0.5_dp	day length fraction
procedure(decl), pointer	daylength	day length function
procedure(decl), pointer	declination	declination function
real(dp), dimension(:), allocatable	dil_din	
real(dp), dimension(:), allocatable	dil_dip	
real(dp), dimension(:), allocatable	dil_fact	
real(dp), dimension(:), allocatable	dil_time	
logical	dilute =.FALSE.	
integer	doy	day of year
real(dp)	dto =1D0	output time step (not related to time step for integration)
integer	dtutc	difference between local time and UTC in s
real(dp)	end	time of end of simulation
real(dp)	hour	current hour of the day
real(dp)	latdeg	
real(dp), pointer	latrad	
real(dp)	londeg	
integer	ndil =0	number of dilution events
real(dp)	noon =12._dp* cfhs	noon-time of day in s
real(dp)	now	current model time in s in the main program
real(dp)	previous	model time at previous output
real(dp)	ptl =0.8333_dp	twilight parameter in degrees
real(dp)	sample =0D0	
real(dp)	start =0._dp	time of start of simulation
integer(int64), dimension(4)	sun	current-day sunrise, noon, sunset, midnight in s
procedure(sunday), pointer	sunday	
integer(kind=int64), pointer	sunrise	
integer(kind=int64), pointer	sunset	
integer(int64)	tbase	time 0 in s with respect to date units in forcing file
integer	timeout =0	

Class Members

real(dp)	<code>tin =0D0</code>	time of initial conditions in initial-conditions file
real(dp)	<code>toy</code>	time of year in decimal days
real(dp)	<code>w1</code>	sunset hour angle

5.9.3 Function/Subroutine Documentation

5.9.3.1 `ft_eppley()`

```
real(dp) function et::ft_eppley (
    class(funigroup), intent(in) grp,
    type(layer), intent(in) box )
```

Temperature function after Eppley (1972)

Definition at line [295](#) of file [et.f90](#).

Referenced by [ft_select\(\)](#).

Here is the caller graph for this function:



5.9.3.2 `ft_houlton()`

```
real(dp) function et::ft_houlton (
    class(funigroup), intent(in) grp,
    type(layer), intent(in) box )
```

Temperature function after Houlton et al. (2008)

Definition at line [303](#) of file [et.f90](#).

Referenced by [ft_select\(\)](#).

Here is the caller graph for this function:



5.9.3.3 ft_me()

```
real(dp) function et::ft_me (  
    class(fungroup), intent(in) grp,  
    type(layer), intent(in) box )
```

Temperature function for Mytilus edulis.

Definition at line 320 of file [et.f90](#).

Referenced by [ft_select\(\)](#).

Here is the caller graph for this function:



5.9.3.4 ft_oppla()

```
real(dp) function et::ft_oppla (  
    class(fungroup), intent(in) grp,  
    type(layer), intent(in) box )
```

Temperature function based on Moisan et al. (2002)

Definition at line 311 of file [et.f90](#).

Referenced by [ft_select\(\)](#).

Here is the caller graph for this function:



5.9.3.5 ft_select()

```
subroutine et::ft_select (  
    class(fungroup), intent(inout) grp,  
    character(len=*), intent(in) ft,  
    character(len=*), intent(in) caller )
```

Select and associate temperature-dependence function.

Parameters

<code>in, out</code>	<i>grp</i>	functional group
<code>in</code>	<i>caller</i>	name of caller (used for error messages)

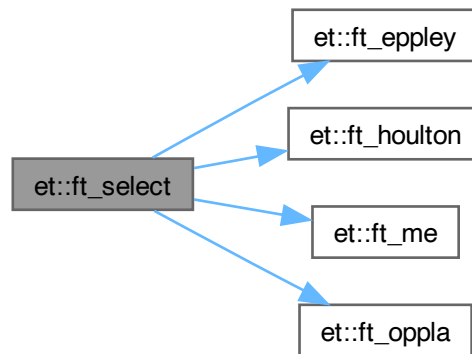
Parameters

<i>ft</i>	name of temperature function ('oppla', 'Eppley', 'Houlton', or 'ME')
-----------	--

Definition at line 328 of file [et.f90](#).

References [ft_eppley\(\)](#), [ft_houlton\(\)](#), [ft_me\(\)](#), and [ft_oppla\(\)](#).

Here is the call graph for this function:



5.9.4 Variable Documentation

5.9.4.1 constituent_units

```
character(len=*), dimension(*), parameter et::constituent_units = (/ 'mmol m-3 ', 'mmol m-3 ',
'mmol m-3 ', 'mmol m-3 ', 'mgChl m-3' /)
```

Definition at line 29 of file [et.f90](#).

5.9.4.2 constituents

```
character(len=*), dimension(*), parameter et::constituents = (/ 'C ', 'N ', 'P ', 'PIC', 'Chl' /)
```

Definition at line 29 of file [et.f90](#).

Referenced by [plankton::plankton_init\(\)](#).

5.9.4.3 pi

```
real(dp), parameter et::pi = 3.14159265358979323_dp
```

Common components for all functional groups.

- Properties common to all groups, e.g., all have a name and track one or more constituents (C, N, P, ...).
- TYPE(group) is extended in the functional-group modules bac, cmo, cfo, etc., to declare the actual types of the functional groups which are then associated with pointer var in variable fungroup (module plankton) of TYPE(state)
- The central part of this type are the deferred procedure pointers flux, read, and set, which must be associated (over-ridden) in the functional-group modules.

Definition at line 28 of file [et.f90](#).

Referenced by [brock81::daylength\(\)](#), [brock81::declination_f95\(\)](#), and [brock81::rdrad\(\)](#).

5.9.4.4 rad

```
real(dp), parameter et::rad = pi/180._dp
```

Definition at line 28 of file [et.f90](#).

Referenced by [brock81::brock81_init\(\)](#).

5.10 fudu Module Reference

FORTRAN interface for the UDUNITS2 library.

Data Types

- interface [cv_convert](#)
- interface [cv_free](#)
- type [quantity](#)
Type for keeping values and units together; enables easy input of units in namelists. [More...](#)
- interface [ut_decode_time](#)
- interface [ut_encode_time](#)
- interface [ut_free](#)
- interface [ut_free_system](#)
- interface [ut_get_converter](#)
- interface [ut_get_status](#)
- interface [ut_parse](#)
Parse unit string.
- interface [ut_read_xml](#)
Set-up and return units system from xml database.

Functions/Subroutines

- `real(c_double)` function, public `convert` (qty, unit)
Convert a quantity to another unit.
- subroutine, public `exudu`
Unregister the units database associated with the global variable `utsystem`.
- subroutine, public `inudu`
Initialise the UDUNITS2 library and read the units database.
- subroutine, public `slot` (us1, us2, slope, offset)
Slope and offset for unit conversion.
- subroutine `uduerr` (udfun, unit, unit2)

Variables

- `integer(c_int)`, parameter `ascii` =0
- `integer(kind=int64)`, parameter, public `cfds` =24*`cfhs`
- `integer`, parameter, public `cfhs` =3600
- `integer(c_int)`, parameter `latin1` =1
- `integer(c_int)`, parameter `ute_bad_arg` =1
- `integer(c_int)`, parameter `ute_cant_format` =9
- `integer(c_int)`, parameter `ute_exists` =2
- `integer(c_int)`, parameter `ute_meaningless` =6
- `integer(c_int)`, parameter `ute_no_second` =7
- `integer(c_int)`, parameter `ute_no_unit` =3
- `integer(c_int)`, parameter `ute_not_same_system` =5
- `integer(c_int)`, parameter `ute_open_arg` =12
- `integer(c_int)`, parameter `ute_open_default` =14
- `integer(c_int)`, parameter `ute_open_env` =13
- `integer(c_int)`, parameter `ute_os` =4
- `integer(c_int)`, parameter `ute_parse` =15
- `integer(c_int)`, parameter `ute_success` =0
- `integer(c_int)`, parameter `ute_syntax` =10
- `integer(c_int)`, parameter `ute_unknown` =11
- `integer(c_int)`, parameter `ute_visit_error` =8
- `integer(c_int)`, private `utencoding`
- `integer(c_int)`, parameter `utf8` =2
- `integer(c_intptr_t)`, private `utsystem`

5.10.1 Detailed Description

FORTTRAN interface for the UDUNITS2 library.

see the [UDUNITS2 documentation](#) for details of the interfaced C functions

5.10.2 Data Type Documentation

5.10.2.1 type `fudu::quantity`

Type for keeping values and units together; enables easy input of units in namelists.

Definition at line 14 of file `fudu.F90`.

Class Members

character(kind=c_char, len=250)	unit	units string
real(c_double)	value	numeric value

5.10.3 Function/Subroutine Documentation

5.10.3.1 convert()

```
real(c_double) function, public fudu::convert (
    type(quantity), intent(in) qty,
    character(kind=c_char, len=*), intent(in) unit )
```

Convert a quantity to another unit.

Parameters

in	<i>qty</i>	quantity to be converted
in	<i>unit</i>	target units

Returns

value of quantity in target units

Definition at line 113 of file [fudu.F90](#).

References [uduer\(\)](#), [utencoding](#), and [utsystem](#).

Here is the call graph for this function:



5.10.3.2 exudu()

```
subroutine, public fudu::exudu
```

Unregister the units database associated with the global variable utsystem.

Definition at line 108 of file [fudu.F90](#).

References [utsystem](#).

5.10.3.3 inudu()

```
subroutine, public fudu::inudu
```

Initialise the UDUNITS2 library and read the units database.

Definition at line 96 of file [fudu.F90](#).

References [uduer\(\)](#), [utencoding](#), [utf8](#), and [utsystem](#).

Here is the call graph for this function:



5.10.3.4 slot()

```
subroutine, public fudu::slot (
    character(kind=c_char,len=*), intent(in) us1,
    character(kind=c_char,len=*), intent(in) us2,
    real(c_double), intent(out) slope,
    real(c_double), intent(out), optional offset )
```

Slope and offset for unit conversion.

A quantity q_{us1} in units given by units string $us1$ can be converted to units given by $us2$ by

$$q_{us2} = \text{offset} + \text{slope} \cdot q_{us1} \quad (5.1)$$

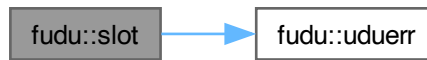
Parameters

in	<i>us1</i>	original units string
in	<i>us2</i>	target units string
out	<i>slope</i>	slope of units conversion
out	<i>offset</i>	offset of units conversion

Definition at line 138 of file [fudu.F90](#).

References [uduer\(\)](#), [utencoding](#), and [utsystem](#).

Here is the call graph for this function:



5.10.3.5 uduerr()

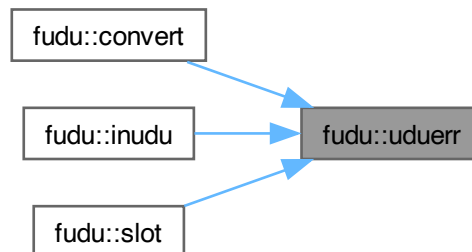
```

subroutine fudu::udurr (
    character(len=*), intent(in) udfun,
    character(len=*), intent(in), optional unit,
    character(len=*), intent(in), optional unit2 ) [private]
  
```

Definition at line 157 of file [fudu.F90](#).

Referenced by [convert\(\)](#), [inudu\(\)](#), and [slot\(\)](#).

Here is the caller graph for this function:



5.10.4 Variable Documentation

5.10.4.1 ascii

```

integer(c_int), parameter fudu::ascii =0 [private]
  
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.2 cfds

```

integer(kind=int64), parameter, public fudu::cfds =24*cfhs
  
```

Definition at line 19 of file [fudu.F90](#).

5.10.4.3 cfhs

```
integer, parameter, public fudu::cfhs =3600
```

Definition at line 18 of file [fudu.F90](#).

5.10.4.4 latin1

```
integer(c_int), parameter fudu::latin1 =1 [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.5 ute_bad_arg

```
integer(c_int), parameter fudu::ute_bad_arg =1 [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.6 ute_cant_format

```
integer(c_int), parameter fudu::ute_cant_format =9 [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.7 ute_exists

```
integer(c_int), parameter fudu::ute_exists =2 [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.8 ute_meaningless

```
integer(c_int), parameter fudu::ute_meaningless =6 [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.9 ute_no_second

```
integer(c_int), parameter fudu::ute_no_second =7 [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.10 ute_no_unit

```
integer(c_int), parameter fudu::ute_no_unit =3 [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.11 ute_not_same_system

```
integer(c_int), parameter fudu::ute_not_same_system =5  [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.12 ute_open_arg

```
integer(c_int), parameter fudu::ute_open_arg =12  [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.13 ute_open_default

```
integer(c_int), parameter fudu::ute_open_default =14  [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.14 ute_open_env

```
integer(c_int), parameter fudu::ute_open_env =13  [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.15 ute_os

```
integer(c_int), parameter fudu::ute_os =4  [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.16 ute_parse

```
integer(c_int), parameter fudu::ute_parse =15  [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.17 ute_success

```
integer(c_int), parameter fudu::ute_success =0  [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.18 ute_syntax

```
integer(c_int), parameter fudu::ute_syntax =10  [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.19 ute_unknown

```
integer(c_int), parameter fudu::ute_unknown =11 [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.20 ute_visit_error

```
integer(c_int), parameter fudu::ute_visit_error =8 [private]
```

Definition at line 20 of file [fudu.F90](#).

5.10.4.21 utencoding

```
integer(c_int), private fudu::utencoding [private]
```

Definition at line 25 of file [fudu.F90](#).

Referenced by [convert\(\)](#), [inudu\(\)](#), and [slot\(\)](#).

5.10.4.22 utf8

```
integer(c_int), parameter fudu::utf8 =2 [private]
```

Definition at line 20 of file [fudu.F90](#).

Referenced by [inudu\(\)](#).

5.10.4.23 utsystem

```
integer(c_intptr_t), private fudu::utsystem [private]
```

Definition at line 24 of file [fudu.F90](#).

Referenced by [convert\(\)](#), [exudu\(\)](#), [inudu\(\)](#), and [slot\(\)](#).

5.11 julian Module Reference

Fortran implementation of the [julian library](#).

Data Types

- interface [c_strftime](#)
Interface to POSIX `strftime`. Returns a formatted time string, given input time struct and format. See <https://www.cplusplus.com/reference/ctime/strftime> for reference.
- interface [c_strptime](#)
Interface to POSIX `strptime`. Returns a time struct object based on the input time string `str` and format. See <https://man7.org/linux/man-pages/man3/strptime.3.html> for reference.
- type [leap](#)
- type [tm_struct](#)
Derived type for compatibility with C and C++ struct `tm`. [More...](#)
- type [yms](#)

Functions/Subroutines

- subroutine, public [jul_dhmsfsec](#) (secs, day, hour, minute, seconds)
convert number of seconds to days, hours, minutes and seconds
- subroutine [jul_dsofsec](#) (secs, day, seconds)
- integer function [jul_dutcofymd](#) (year, month, day)
- subroutine [jul_fixym](#) (year, month)
- character(len=200) function, public [jul_formatdate](#) (dutc, seconds, fmt)
Format a date and time according to a format string.
- subroutine [jul_gregymdofjd](#) (jd, year, month, day)
- subroutine [jul_initleaps](#)
- logical function [jul_isleapday](#) (dutc)
Determines whether a given day contains a leap second.
- integer function [jul_jdofgregymd](#) (year, month, day)
- integer function [jul_jdofjulymd](#) (year, month, day)
- subroutine [jul_julymdofjd](#) (jd, year, month, day)
- integer function [jul_leapsecs](#) (dutc)
number of leap seconds elapsed before a given day
- integer function [jul_leapsecsym](#) (year, month)
- subroutine, public [jul_parsedt](#) (string, fmt, dutc, secs)
Interprets a character string as a date and time.
- real(dp) function [jul_taiofdutc](#) (dutc)
This internal function calculates the number of seconds TAI from the floating-point number of days UTC relative to noon J2000.
- real(dp) function [jul_taiofet](#) (et)
Convert from ET (ephemeris time) to TAI (atomic time).
- real(dp) function [jul_taiofjd](#) (jd, type)
Convert Julian date to a number of seconds relative to J2000 TAI.
- subroutine, public [jul_ydofdutc](#) (dutc, year, doy)
Returns the calendar year and day-of-year number for a day number relative to January 1, 2000 for a given date.
- subroutine, public [jul_ymdofdutc](#) (dutc, year, month, day)
Returns the calendar year, month and day for a day number relative to January 1, 2000 for a given date.
- integer function [leap_index](#) (year, month)

Variables

- integer, parameter `dp` =SELECTED_REAL_KIND(KIND(0D0))
- integer, parameter `gregorian_day` =15
- integer, parameter `gregorian_dutc` =-152384
- integer, parameter `gregorian_month` =10
- integer, parameter `gregorian_year` =1582
- real(`dp`), parameter `jd_of_j2000_noon` =2451545.0_dp
- integer, parameter `jdn_of_j2000` =2451545
Julian day numer of 1 January 2000, i.e., since 4713-01-01 BCE = -4712-01-01.
- integer, parameter `jul_et_type` =2
- integer, parameter `jul_tai_type` =1
- integer, parameter `jul_utc_type` =0
- type(`yms`), dimension(*), parameter `leap_defaults` =(/ `yms`(1972, 1, 10), `yms`(1972, 7, 11), `yms`(1973, 1, 12), `yms`(1974, 1, 13), `yms`(1975, 1, 14), `yms`(1976, 1, 15), `yms`(1977, 1, 16), `yms`(1978, 1, 17), `yms`(1979, 1, 18), `yms`(1980, 1, 19), `yms`(1981, 7, 20), `yms`(1982, 7, 21), `yms`(1983, 7, 22), `yms`(1985, 7, 23), `yms`(1988, 1, 24), `yms`(1990, 1, 25), `yms`(1991, 1, 26), `yms`(1992, 7, 27), `yms`(1993, 7, 28), `yms`(1994, 7, 29), `yms`(1996, 1, 30), `yms`(1997, 7, 31), `yms`(1999, 1, 32), `yms`(2006, 1, 33), `yms`(2009, 1, 34), `yms`(2012, 7, 35), `yms`(2015, 7, 36), `yms`(2017, 1, 37)/)
- integer, dimension(:), allocatable `leap_table`
leap-seconds table
- type(`leap`) `leaps`
- real(`dp`), parameter `mjd_of_j2000_noon` =51544.5_dp

5.11.1 Detailed Description

Fortran implementation of the `julian library`.

The main differences are that the conversion routines from/to character strings (`Jul_FormatDate` and `Jul_ParseDT`) are replaced by wrappers around `strptime` and `strftime` and seconds are mostly represented as integers.

5.11.2 Data Type Documentation

5.11.2.1 type `julian::leap`

Definition at line 23 of file `julian.f90`.

Class Members

integer	<code>nmax</code>	number of leap seconds as of <code>ymax</code>
integer	<code>nmin</code>	number of leap second before <code>ymin</code> = 1972
integer	<code>size</code>	size of leap-seconds table
integer	<code>ymax</code>	last year of leap-seconds table
integer	<code>ymin</code>	first year of leap-seconds table (1972)

5.11.2.2 type `julian::tm_struct`

Derived type for compatibility with C and C++ struct `tm`.

Enables calling `strptime` and `strftime` using `iso_c_binding`. See <https://www.cplusplus.com/reference/ctime/tm> for reference.

Definition at line 35 of file [julian.f90](#).

Class Members

<code>integer(c_int)</code>	<code>tm_hour = 0</code>	Hours [0–23].
<code>integer(c_int)</code>	<code>tm_isdst = 0</code>	DST [–1/0/1].
<code>integer(c_int)</code>	<code>tm_mday = 0</code>	Day [1–31].
<code>integer(c_int)</code>	<code>tm_min = 0</code>	Minutes [0–59].
<code>integer(c_int)</code>	<code>tm_mon = 0</code>	Month [0–11].
<code>integer(c_int)</code>	<code>tm_sec = 0</code>	Seconds [0–60] (1 leap second)
<code>integer(c_int)</code>	<code>tm_wday = 0</code>	Day of week [0–6].
<code>integer(c_int)</code>	<code>tm_yday = 0</code>	Days in year [0–365].
<code>integer(c_int)</code>	<code>tm_year = 0</code>	Year – 1900.

5.11.2.3 type `julian::yms`

Definition at line 18 of file [julian.f90](#).

Class Members

<code>integer</code>	<code>month</code>	month for leap second
<code>integer</code>	<code>seconds</code>	cumulative leap seconds at month, year
<code>integer</code>	<code>year</code>	year for leap second

5.11.3 Function/Subroutine Documentation

5.11.3.1 `jul_dhmsfsec()`

```
subroutine, public julian::jul_dhmsfsec (
    integer, intent(in) secs,
    integer, intent(out) day,
    integer, intent(out) hour,
    integer, intent(out) minute,
    integer, intent(out) seconds )
```

convert number of seconds to days, hours, minutes and seconds

This function converts a number of seconds to days, hours, minutes and seconds. All days are assumed to contain 86400 seconds; leap seconds are not supported.

Parameters

<code>in</code>	<code>secs</code>	number of seconds
<code>out</code>	<code>seconds</code>	number of seconds into minute (0–60)

Parameters

<i>day</i>	number of days
<i>hour</i>	number of hours into day (0--23)
<i>minute</i>	number of minutes into hour (0--59)

Definition at line 345 of file [julian.f90](#).

References [jul_dsofsec\(\)](#).

Here is the call graph for this function:



5.11.3.2 jul_dsofsec()

```

subroutine julian::jul_dsofsec (
    integer, intent(in) secs,
    integer, intent(out) day,
    integer, intent(out) seconds ) [private]
  
```

Definition at line 359 of file [julian.f90](#).

Referenced by [jul_dhmsofsec\(\)](#).

Here is the caller graph for this function:



5.11.3.3 jul_dutcofynd()

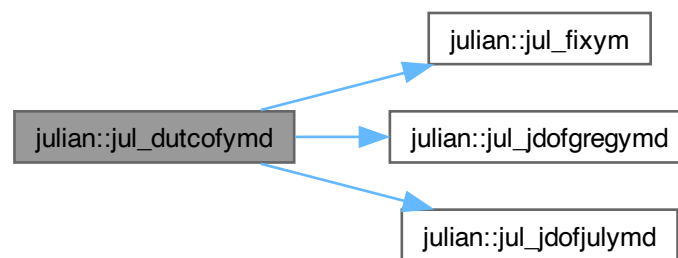
```
integer function julian::jul_dutcofynd (
    integer, intent(inout) year,
    integer, intent(inout) month,
    integer, intent(in) day ) [private]
```

Definition at line 209 of file [julian.f90](#).

References [gregorian_day](#), [gregorian_month](#), [gregorian_year](#), [jdn_of_j2000](#), [jul_fixym\(\)](#), [jul_jdofgregymd\(\)](#), and [jul_jdofjulymd\(\)](#).

Referenced by [jul_parsedt\(\)](#), and [jul_ydofdutc\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.3.4 jul_fixym()

```
subroutine julian::jul_fixym (
    integer, intent(inout) year,
    integer, intent(inout) month ) [private]
```

Definition at line 302 of file [julian.f90](#).

Referenced by [jul_dutcofynd\(\)](#).

Here is the caller graph for this function:



5.11.3.5 jul_formatdate()

```
character(len=200) function, public julian::jul_formatdate (
    integer, intent(in) dutc,
    integer, intent(in) seconds,
    character(len=*), intent(in) fmt )
```

Format a date and time according to a format string.

This is a wrapper function for the `strftime` C function.

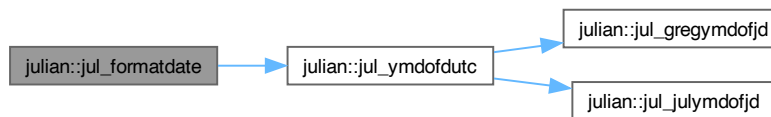
Returns

formatted string

Definition at line 396 of file `julian.f90`.

References `jul_ymdofdutc()`.

Here is the call graph for this function:



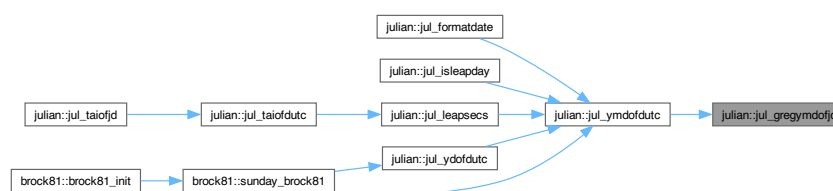
5.11.3.6 jul_gregymdofjd()

```
subroutine julian::jul_gregymdofjd (
    integer, intent(in) jd,
    integer, intent(out) year,
    integer, intent(out) month,
    integer, intent(out) day ) [private]
```

Definition at line 270 of file `julian.f90`.

Referenced by `jul_ymdofdutc()`.

Here is the caller graph for this function:



5.11.3.7 jul_initleaps()

subroutine julian::jul_initleaps [private]

Definition at line 313 of file [julian.f90](#).

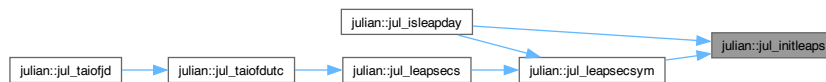
References [leap_defaults](#), [leap_index\(\)](#), [leap_table](#), and [leaps](#).

Referenced by [jul_isleapday\(\)](#), and [jul_leapsecsym\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.3.8 jul_isleapday()

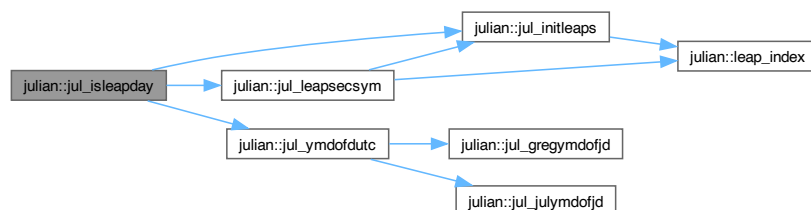
logical function julian::jul_isleapday (
integer, intent(in) dutc) [private]

Determines whether a given day contains a leap second.

Definition at line 191 of file [julian.f90](#).

References [jul_initleaps\(\)](#), [jul_leapsecsym\(\)](#), [jul_ymdofdutc\(\)](#), and [leap_table](#).

Here is the call graph for this function:



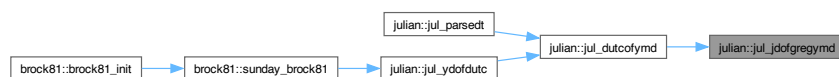
5.11.3.9 jul_jdofgregymd()

```
integer function julian::jul_jdofgregymd (
    integer, intent(in) year,
    integer, intent(in) month,
    integer, intent(in) day ) [private]
```

Definition at line 223 of file [julian.f90](#).

Referenced by [jul_dutcofymd\(\)](#).

Here is the caller graph for this function:



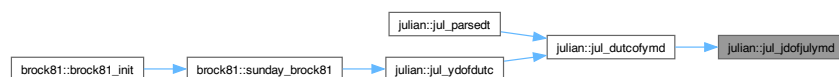
5.11.3.10 jul_jdofjulymd()

```
integer function julian::jul_jdofjulymd (
    integer, intent(in) year,
    integer, intent(in) month,
    integer, intent(in) day ) [private]
```

Definition at line 232 of file [julian.f90](#).

Referenced by [jul_dutcofymd\(\)](#).

Here is the caller graph for this function:



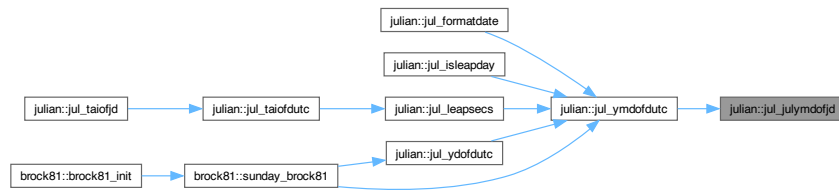
5.11.3.11 jul_julymdofjd()

```
subroutine julian::jul_julymdofjd (
    integer, intent(in) jd,
    integer, intent(out) year,
    integer, intent(out) month,
    integer, intent(out) day ) [private]
```

Definition at line 286 of file [julian.f90](#).

Referenced by [jul_ymdofdutc\(\)](#).

Here is the caller graph for this function:



5.11.3.12 jul_leapsecs()

```
integer function julian::jul_leapsecs (
    integer, intent(in) dutc ) [private]
```

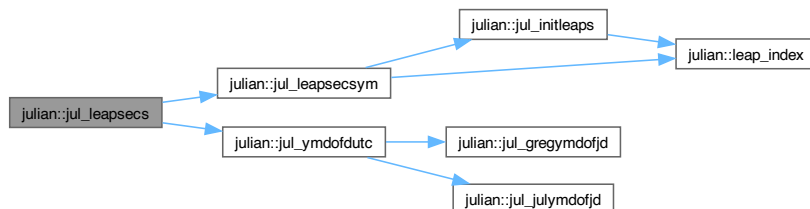
number of leap seconds elapsed before a given day

Definition at line 163 of file [julian.f90](#).

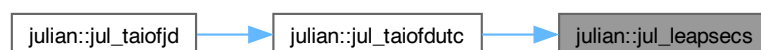
References [jul_leapsecsym\(\)](#), and [jul_ymdofdutc\(\)](#).

Referenced by [jul_taiofdutc\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.3.13 jul_leapsecsym()

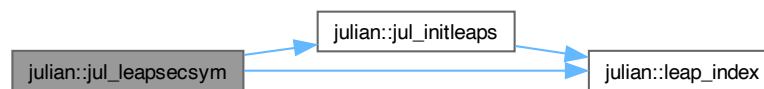
```
integer function julian::jul_leapsecsym (
    integer, intent(in) year,
    integer, intent(in) month ) [private]
```

Definition at line 176 of file [julian.f90](#).

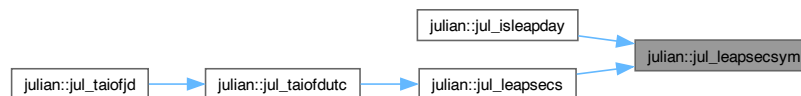
References [jul_initleaps\(\)](#), [leap_index\(\)](#), [leap_table](#), and [leaps](#).

Referenced by [jul_isleapday\(\)](#), and [jul_leapsecs\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.3.14 jul_parsedt()

```
subroutine, public julian::jul_parsedt (
    character(len=*), intent(in) string,
    character(len=*), intent(in) fmt,
    integer, intent(out) dutc,
    integer, intent(out) secs )
```

Interprets a character string as a date and time.

This is a wrapper for the `strptime` C function.

Parameters

in	<i>fmt</i>	format string
out	<i>secs</i>	inferred seconds into day

Parameters

<i>string</i>	the character string to parse
<i>dutc</i>	inferred days relative to January 1, 2000

Definition at line 374 of file [julian.f90](#).

References [jul_dutcofynd\(\)](#).

Here is the call graph for this function:

5.11.3.15 `jul_taiofdutc()`

```

real(dp) function julian::jul_taiofdutc (
    real(dp), intent(in) dutc ) [private]

```

This internal function calculates the number of seconds TAI from the floating-point number of days UTC relative to noon J2000.

Returns

seconds from J2000 noon TAI

Parameters

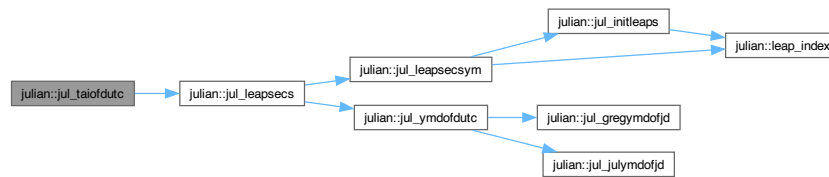
<i>in</i>	<i>dutc</i>	days relative to J2000 noon UTC
-----------	-------------	---------------------------------

Definition at line 150 of file [julian.f90](#).

References [jul_leapsecs\(\)](#), and [leaps](#).

Referenced by [jul_taiofjd\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.3.16 jul_taiofet()

```

real(dp) function julian::jul_taiofet (
    real(dp), intent(in) et ) [private]

```

Convert from ET (ephemeris time) to TAI (atomic time).

This function converts from ET (ephemeris time) to TAI (Atomic time). All times are given as seconds relative to J2000. The algorithm is based on that used in the SPICE toolkit.

Returns

corresponding atomic time (seconds relative to J2000 TAI)

Parameters

<code>in</code>	<code>et</code>	ephemeris time
-----------------	-----------------	----------------

Definition at line 136 of file [julian.f90](#).

Referenced by [jul_taiofjd\(\)](#).

Here is the caller graph for this function:



5.11.3.17 jul_taiofjd()

```

real(dp) function julian::jul_taiofjd (
    real(dp), intent(in) jd,
    integer, intent(in) type ) [private]
  
```

Convert Julian date to a number of seconds relative to J2000 TAI.

This function converts a Julian date to a number of seconds relative to J2000 TAI. The Julian date may be given in any of three possible time systems: Universal time (UTC), Atomic time (TAI), or Ephemeris time (ET).

If the date type is UTC, fractional days are scaled by the number of seconds between one UTC noon and the next; since some days have leap seconds, this makes the spacing of UTC Julian dates slightly non-uniform.

Returns

number of seconds since noon J2000 TAI

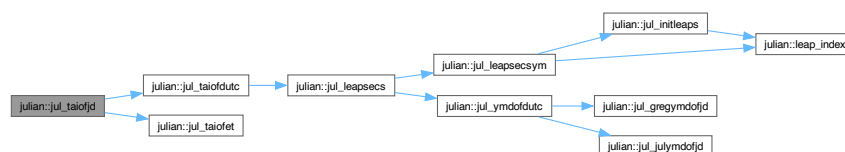
Parameters

in	<i>jd</i>	Julian date, in one of three possible time systems
in	<i>type</i>	type of the Julian date, one of the constants JUL_UTC_TYPE (0), JUL_TAI_TYPE (1), or JUL_ET_TYPE (2)

Definition at line 113 of file [julian.f90](#).

References [jd_of_j2000_noon](#), [jul_et_type](#), [jul_tai_type](#), [jul_taiofdutc\(\)](#), and [jul_taiofet\(\)](#).

Here is the call graph for this function:



5.11.3.18 jul_ydofdutc()

```
subroutine, public julian::jul_ydofdutc (
    integer, intent(in) dutc,
    integer, intent(out) year,
    integer, intent(out) doy )
```

Returns the calendar year and day-of-year number for a day number relative to January 1, 2000 for a given date.

This subroutine returns the calendar year and day-of-year for a day number relative to January 1, 2000.

Parameters

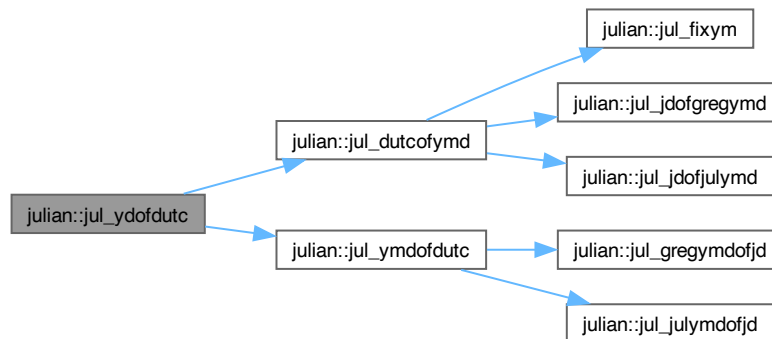
in	<i>dutc</i>	number of days relative to January 1, 2000
out	<i>year</i>	year value
out	<i>doy</i>	day-of-year (1-366)

Definition at line 244 of file [julian.f90](#).

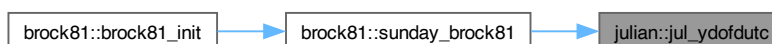
References [jul_dutcofynd\(\)](#), and [jul_ymdofdutc\(\)](#).

Referenced by [brock81::sunday_brock81\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.3.19 jul_ymdofdutc()

```

subroutine, public julian::jul_ymdofdutc (
    integer, intent(in) dutc,
    integer, intent(out) year,
    integer, intent(out) month,
    integer, intent(out) day )

```

Returns the calendar year, month and day for a day number relative to January 1, 2000 for a given date.

This subroutine returns the calendar year, month and day for a day number relative to January 1, 2000.

Parameters

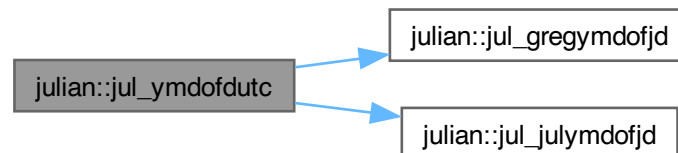
in	<i>dutc</i>	number of days relative to January 1, 2000
out	<i>year</i>	calendar year
out	<i>month</i>	month (1-12)
out	<i>day</i>	day (1-31)

Definition at line 258 of file [julian.f90](#).

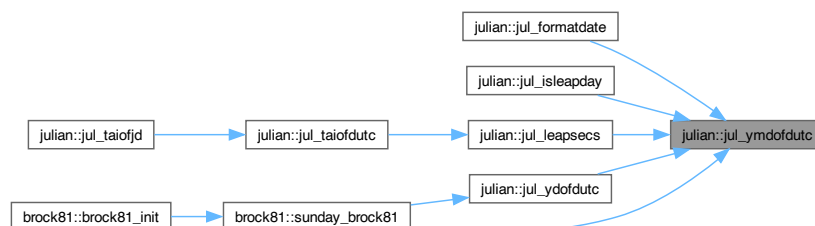
References [gregorian_dutc](#), [jdn_of_j2000](#), [jul_gregymdofjd\(\)](#), and [jul_julymdofjd\(\)](#).

Referenced by [jul_formatdate\(\)](#), [jul_isleapday\(\)](#), [jul_leapsecs\(\)](#), [jul_ydofdutc\(\)](#), and [brock81::sunday_brock81\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.3.20 leap_index()

```
integer function julian::leap_index (
    integer, intent(in) year,
    integer, intent(in) month ) [private]
```

Definition at line 170 of file [julian.f90](#).

References [leaps](#).

Referenced by [jul_initleaps\(\)](#), and [jul_leapsecsym\(\)](#).

Here is the caller graph for this function:



5.11.4 Variable Documentation

5.11.4.1 dp

```
integer, parameter julian::dp =SELECTED_REAL_KIND(KIND(0D0)) [private]
```

Definition at line 12 of file [julian.f90](#).

5.11.4.2 gregorian_day

```
integer, parameter julian::gregorian_day =15 [private]
```

Definition at line 12 of file [julian.f90](#).

Referenced by [jul_dutcofynd\(\)](#).

5.11.4.3 gregorian_dutc

```
integer, parameter julian::gregorian_dutc =-152384 [private]
```

Definition at line 12 of file [julian.f90](#).

Referenced by [jul_ymdofdutc\(\)](#).

5.11.4.4 gregorian_month

```
integer, parameter julian::gregorian_month =10 [private]
```

Definition at line 12 of file [julian.f90](#).

Referenced by [jul_dutcofynd\(\)](#).

5.11.4.5 gregorian_year

```
integer, parameter julian::gregorian_year =1582 [private]
```

Definition at line 12 of file [julian.f90](#).

Referenced by [jul_dutcofynd\(\)](#).

5.11.4.6 jd_of_j2000_noon

```
real(dp), parameter julian::jd_of_j2000_noon =2451545.0_dp [private]
```

Definition at line 17 of file [julian.f90](#).

Referenced by [jul_taiofjd\(\)](#).

5.11.4.7 jdn_of_j2000

```
integer, parameter julian::jdn_of_j2000 =2451545 [private]
```

Julian day numer of 1 January 2000, i.e., since 4713-01-01 BCE = −4712-01-01.

Definition at line 12 of file [julian.f90](#).

Referenced by [jul_dutcofynd\(\)](#), and [jul_ymdofdutc\(\)](#).

5.11.4.8 jul_et_type

```
integer, parameter julian::jul_et_type =2 [private]
```

Definition at line 12 of file [julian.f90](#).

Referenced by [jul_taiofjd\(\)](#).

5.11.4.9 jul_tai_type

```
integer, parameter julian::jul_tai_type =1 [private]
```

Definition at line 12 of file [julian.f90](#).

Referenced by [jul_taiofjd\(\)](#).

5.11.4.10 jul_utc_type

```
integer, parameter julian::jul_utc_type =0 [private]
```

Definition at line 12 of file [julian.f90](#).

5.11.4.11 leap_defaults

```
type(yms), dimension(*), parameter julian::leap_defaults =(/ yms(1972, 1, 10), yms(1972, 7, 11), yms(1973, 1, 12), yms(1974, 1, 13), yms(1975, 1, 14), yms(1976, 1, 15), yms(1977, 1, 16), yms(1978, 1, 17), yms(1979, 1, 18), yms(1980, 1, 19), yms(1981, 7, 20), yms(1982, 7, 21), yms(1983, 7, 22), yms(1985, 7, 23), yms(1988, 1, 24), yms(1990, 1, 25), yms(1991, 1, 26), yms(1992, 7, 27), yms(1993, 7, 28), yms(1994, 7, 29), yms(1996, 1, 30), yms(1997, 7, 31), yms(1999, 1, 32), yms(2006, 1, 33), yms(2009, 1, 34), yms(2012, 7, 35), yms(2015, 7, 36), yms(2017, 1, 37)/)
```

Definition at line 46 of file [julian.f90](#).

Referenced by [jul_initleaps\(\)](#).

5.11.4.12 leap_table

```
integer, dimension(:), allocatable julian::leap_table [private]
```

leap-seconds table

Definition at line 76 of file [julian.f90](#).

Referenced by [jul_initleaps\(\)](#), [jul_isleapday\(\)](#), and [jul_leapsecsym\(\)](#).

5.11.4.13 leaps

```
type(leap) julian::leaps [private]
```

Definition at line 75 of file [julian.f90](#).

Referenced by [jul_initleaps\(\)](#), [jul_leapsecsym\(\)](#), [jul_taiofdutc\(\)](#), and [leap_index\(\)](#).

5.11.4.14 mjd_of_j2000_noon

```
real(dp), parameter julian::mjd_of_j2000_noon =51544.5_dp [private]
```

Definition at line 17 of file [julian.f90](#).

5.12 lambert Module Reference

F90 module for the Lambert-W function made from TOMS 743.

Data Types

- interface [lambertw](#)

Functions/Subroutines

- elemental real([dp](#)) function [lwm1](#) (x)
lwm1: closed-form approximation of -1-branch of the Lambert-W function
- elemental double precision function [wapd](#) (x, nb, l)
- elemental real function [wapr](#) (x, nb, l)

Variables

- integer, parameter, private [dp](#) =KIND(0D0)

5.12.1 Detailed Description

F90 module for the Lambert-W function made from TOMS 743.

real gets 6 digits right, double precision 15

5.12.2 Function/Subroutine Documentation**5.12.2.1 lwm1()**

```
elemental real(dp) function lambert::lwm1 (  
    real(dp), intent(in) x )
```

[lwm1](#): closed-form approximation of -1-branch of the Lambert-W function

Definition at line [14](#) of file [lambert.f90](#).

Referenced by [cfo::cfo_read\(\)](#).

Here is the caller graph for this function:

**5.12.2.2 wapd()**

```
elemental double precision function lambert::wapd (  
    double precision, intent(in) x,  
    integer, intent(in) nb,  
    integer, intent(in) l )
```

Definition at line [246](#) of file [lambert.f90](#).

5.12.2.3 wapr()

```
elemental real function lambert::wapr (
    real, intent(in) x,
    integer, intent(in) nb,
    integer, intent(in) l )
```

Definition at line 28 of file [lambert.f90](#).

5.12.3 Variable Documentation

5.12.3.1 dp

```
integer, parameter, private lambert::dp =KIND(0D0) [private]
```

Definition at line 6 of file [lambert.f90](#).

5.13 onf Module Reference

NetCDF interface for oppla.

Data Types

- type [dataset](#)
- type [dimension](#)
- type [nf90grd](#)
- type [nf90group](#)
- type [nf90stratt](#)
- type [nf90var](#)
- type [nf90vec](#)
- type [phydat](#)
 physical data
- interface [rpd](#)
- type [statevars](#)

Functions/Subroutines

- subroutine [exof](#) (stv)
 Close output file.
- subroutine [innf](#) (states, phys, lun, atts)
 Initialise NETCDF interface and read associated parameters.
- subroutine [inpd](#) (pdf, times, env, box, svd)
 Open physical data file and read forcing datasets.
- subroutine [invar](#) (stv, ofn, control, dfn, pfn, force, resume)
 Read initial state-variable values and create and initialize output file.
- subroutine [nf90info](#) (group, latdeg, londeg, t0, t1, writeable)
 open groupfilename, read global attributes and coordinates (depth, lon, lat, time)
- subroutine [onf_error](#) (caller, nffun, name, nferr)

- subroutine [outsv](#) (stv, times, env)
Write time in UTC and current state-variable values to file.
- subroutine [read_dim](#) (dim, ncid, caller, nferr)
Load dimension variable.
- subroutine [read_ds](#) (ds, group, data)
read dataset in group described by ds into array data
- subroutine [readpd](#) (pda, time)
Read physical data for time interval encompassing phytim.
- subroutine [write_output](#) (stv, env, times)

Variables

- character(len=14), parameter, public [bfan](#) ='Bottom File'
- character(len=14), parameter, public [dfan](#) ='Data File'
- integer, public [ico2ds](#)
- character(len=14), parameter, public [infan](#) ='Start File'
- integer, public [iprsds](#)
- character(len=14), parameter, public [latan](#) ='latitude'
- character(len=14), parameter, public [lonan](#) ='longitude'
- integer [nbvds](#)
- integer [ngdds](#)
- integer [npfds](#)
- integer [nsfds](#)
- integer [nsvds](#)
- real([dp](#)), [dimension](#)(:,:), allocatable [offsets](#)
- real([dp](#)), [dimension](#)(:,:,:), allocatable, target [pdgds](#)
- real([dp](#)), [dimension](#)(:,:), allocatable, target [pdsds](#)
- character(len=14), parameter, public [pfan](#) ='Parameter File'
- character(len=14), parameter, public [plfan](#) ='Plankton File'
- character(len=14), parameter, public [sufan](#) ='Spin-up File'
- character(len=14), parameter, public [timan](#) ='Time'
- character(len=nf90_max_name), public [timunt](#) ='s'

5.13.1 Detailed Description

NetCDF interface for oppla.

5.13.2 Data Type Documentation

5.13.2.1 type onf::dataset

Definition at line 9 of file [onf.f90](#).

Class Members

character(len=nf90_max_name)	name ="	
character(len=nf90_max_name)	type ="	data type, e.g., Temperature, Salinity, etc.

5.13.2.2 type onf::nf90stratt

Definition at line 13 of file [onf.f90](#).

Class Members

	integer	id	
character(len=nf90_max_name)		name	
character(len=nf90_max_name)		value	

5.13.3 Function/Subroutine Documentation

5.13.3.1 exof()

```
subroutine onf::exof (
    class(statevars), intent(inout) stv ) [private]
```

Close output file.

Parameters

<i>in, out</i>	<i>stv</i>	state-variable structure of the output file
----------------	------------	---

Definition at line 851 of file [onf.f90](#).

Referenced by [onf::statevars::open\(\)](#).

Here is the caller graph for this function:



5.13.3.2 innf()

```
subroutine onf::innf (
    class(statevars), intent(out) states,
    type(phydat), intent(out) phys,
    integer, intent(in) lun,
    type(nf90stratt), dimension(:), intent(out), optional, allocatable atts ) [private]
```

Initialise NETCDF interface and read associated parameters.

Subroutine innf is called as statesopen in plankton

Parameters

out	<i>states</i>	state-variable info
out	<i>phys</i>	info about physical forcing
in	<i>lun</i>	unit number of the control file

Definition at line 107 of file [onf.f90](#).

References [bfan](#), [dfan](#), [infan](#), [nbvds](#), [ngdds](#), [npfds](#), [nsfds](#), [nsvds](#), [pfan](#), [plfan](#), and [sufan](#).

5.13.3.3 inpd()

```
subroutine onf::inpd (
    class(phydat), intent(inout), target pdf,
    type(timing), intent(inout) times,
    type(local), intent(inout) env,
    type(layer), dimension(0:), intent(inout) box,
    type(nf90vec), dimension(:), intent(in) svd ) [private]
```

Open physical data file and read forcing datasets.

Subroutine inpd is called as physinit in plankton:plankton_init

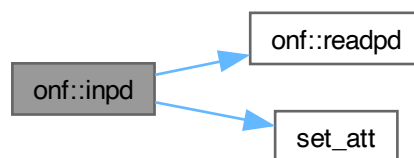
Parameters

in, out	<i>pdf%data</i>	gridded (depth-time) forcing data
in, out	<i>pdf%profile</i>	temporally constant vertical profiles
in, out	<i>pdf%surface</i>	surface data

Definition at line 503 of file [onf.f90](#).

References [nbvds](#), [ngdds](#), [npfds](#), [nsfds](#), [nsvds](#), [pdgds](#), [pdsds](#), [readpd\(\)](#), and [set_att\(\)](#).

Here is the call graph for this function:



5.13.3.4 invar()

```
subroutine onf::invar (
    class(statevars), intent(inout), target stv,
```

```

character(len=*), intent(in) ofn,
character(len=*), intent(in) control,
character(len=*), intent(in) dfn,
character(len=*), intent(in) pfn,
logical, intent(in) force,
logical, intent(in) resume ) [private]

```

Read initial state-variable values and create and initialize output file.

Subroutine `invar` is called as `statesinit` in `plankton:plankton_init`

Parameters

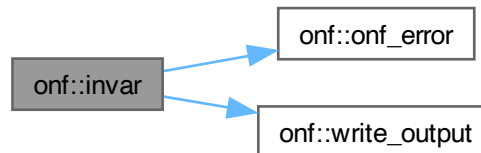
in, out	<i>stv</i>	state-variable structure (type <code>statevars</code>)
in	<i>resume</i>	append output to initial-conditions file
in	<i>force</i>	overwrite output file if it exists
in	<i>ofn</i>	name of output file
out	<i>ystv</i>	allocated <code>real(dp)</code> vector of initial state-variable values

Definition at line 305 of file [onf.f90](#).

References [offsets](#), [onf_error\(\)](#), and [write_output\(\)](#).

Referenced by [onf::statevars::open\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.3.5 nf90info()

```
subroutine onf::nf90info (
    class(nf90group), intent(inout) group,
    real(dp), intent(inout), optional latdeg,
    real(dp), intent(inout), optional londeg,
    real(dp), intent(inout), optional t0,
    real(dp), intent(inout), optional t1,
    logical, intent(in), optional writeable ) [private]
```

open groupfilename, read global attributes and coordinates (depth, lon, lat, time)

Parameters

in	<i>group%depth%name</i>	name of depth dataset
in	<i>group%filename</i>	name of file to read
in	<i>group%lat%name</i>	name of latitude dataset
in	<i>group%lon%name</i>	name of longitude dataset
in	<i>group%time%name</i>	name of time dataset
in	<i>group%timediff</i>	difference between local solar time and UTC in s
out	<i>group%depth%values</i>	depth (z) axis
out	<i>group%lat%values</i>	latitude (y) axis
out	<i>group%depth%values</i>	longitude (x) axis
out	<i>group%time%values</i>	time axis
in, out	<i>t0</i>	start time in s
in, out	<i>t1</i>	end time in s
in	<i>writeable</i>	whether to open the file with read-write access

Definition at line 192 of file [onf.f90](#).

References [onf_error\(\)](#).

Here is the call graph for this function:



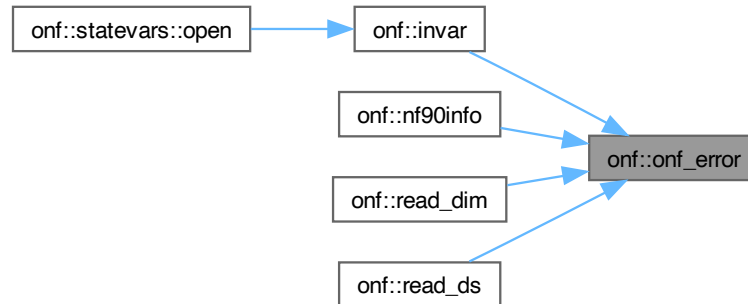
5.13.3.6 onf_error()

```
subroutine onf::onf_error (
    character(len=*), intent(in) caller,
    character(len=*), intent(in) nffun,
    character(len=*), intent(in) name,
    integer, intent(in) nferr ) [private]
```

Definition at line 909 of file [onf.f90](#).

Referenced by [invar\(\)](#), [nf90info\(\)](#), [read_dim\(\)](#), and [read_ds\(\)](#).

Here is the caller graph for this function:



5.13.3.7 outsv()

```

subroutine onf::outsv (
    class(statevars), intent(inout) stv,
    type(timing), intent(in) times,
    type(local), intent(in) env ) [private]

```

Write time in UTC and current state-variable values to file.

Definition at line 813 of file [onf.f90](#).

Referenced by [onf::statevars::open\(\)](#).

Here is the caller graph for this function:



5.13.3.8 read_dim()

```

subroutine onf::read_dim (
    class(dimension), intent(inout) dim,
    integer, intent(in) ncid,
    character(len=*), intent(in) caller,
    integer, intent(out), optional nferr ) [private]

```

Load dimension variable.

Parameters

in	<i>dim</i>	TYPE(dimension) variable to read
in	<i>ncid</i>	id of the file or group of the dataset
in	<i>caller</i>	name of the calling subroutine or function
in	<i>dim%name</i>	name of the dimension
out	<i>dim%values</i>	allocatable array to hold the data
out	<i>dim%varid</i>	variable id of the dimension variable
out	<i>nferr</i>	NetCDF error code

Definition at line 867 of file [onf.f90](#).

References [onf_error\(\)](#).

Here is the call graph for this function:



5.13.3.9 read_ds()

```

subroutine onf::read_ds (
    class(nf90var), intent(inout) ds,
    class(nf90group), intent(in) group,
    real(dp), dimension(*), intent(out) data ) [private]
  
```

read dataset in group described by ds into array data

- if dsunits is set, data will be converted to these units

Definition at line 726 of file [onf.f90](#).

References [onf_error\(\)](#).

Here is the call graph for this function:



5.13.3.10 readpd()

```
subroutine onf::readpd (
    class(phydat), intent(inout) pda,
    real(dp), intent(in) time ) [private]
```

Read physical data for time interval encompassing phytim.

Parameters

in, out	<i>pda</i>	physical data array
in	<i>time</i>	current time

Definition at line 786 of file [onf.f90](#).

References [pdgds](#), and [pdsds](#).

Referenced by [inpd\(\)](#).

Here is the caller graph for this function:



5.13.3.11 write_output()

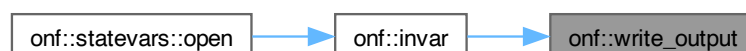
```
subroutine onf::write_output (
    class(statevars), intent(inout) stv,
    type(local), intent(in) env,
    type(timing), intent(in) times ) [private]
```

Definition at line 825 of file [onf.f90](#).

References [offsets](#).

Referenced by [invar\(\)](#).

Here is the caller graph for this function:



5.13.4 Variable Documentation

5.13.4.1 bfan

```
character(len=14), parameter, public onf::bfan ='Bottom File'
```

Definition at line 86 of file [onf.f90](#).

Referenced by [innf\(\)](#).

5.13.4.2 dfan

```
character(len=14), parameter, public onf::dfan ='Data File'
```

Definition at line 86 of file [onf.f90](#).

Referenced by [innf\(\)](#).

5.13.4.3 ico2ds

```
integer, public onf::ico2ds
```

Definition at line 91 of file [onf.f90](#).

5.13.4.4 infan

```
character(len=14), parameter, public onf::infan ='Start File'
```

Definition at line 86 of file [onf.f90](#).

Referenced by [innf\(\)](#).

5.13.4.5 iprds

```
integer, public onf::iprds
```

Definition at line 91 of file [onf.f90](#).

5.13.4.6 latan

```
character(len=14), parameter, public onf::latan ='latitude'
```

Definition at line 86 of file [onf.f90](#).

5.13.4.7 lonan

```
character(len=14), parameter, public onf::lonan = 'longitude'
```

Definition at line 86 of file [onf.f90](#).

5.13.4.8 nbvds

```
integer onf::nbvds
```

Definition at line 85 of file [onf.f90](#).

Referenced by [innf\(\)](#), and [inpd\(\)](#).

5.13.4.9 ngdds

```
integer onf::ngdds [private]
```

Definition at line 85 of file [onf.f90](#).

Referenced by [innf\(\)](#), and [inpd\(\)](#).

5.13.4.10 npfds

```
integer onf::npfds [private]
```

Definition at line 85 of file [onf.f90](#).

Referenced by [innf\(\)](#), and [inpd\(\)](#).

5.13.4.11 nsfds

```
integer onf::nsfds [private]
```

Definition at line 85 of file [onf.f90](#).

Referenced by [innf\(\)](#), and [inpd\(\)](#).

5.13.4.12 nsvds

```
integer onf::nsvds [private]
```

Definition at line 85 of file [onf.f90](#).

Referenced by [innf\(\)](#), and [inpd\(\)](#).

5.13.4.13 offsets

```
real(dp), dimension(:, :), allocatable onf::offsets [private]
```

Definition at line 92 of file [onf.f90](#).

Referenced by [invar\(\)](#), and [write_output\(\)](#).

5.13.4.14 pdgds

```
real(dp), dimension(:, :, :), allocatable, target onf::pdgds [private]
```

Definition at line 93 of file [onf.f90](#).

Referenced by [inpd\(\)](#), and [readpd\(\)](#).

5.13.4.15 pdsds

```
real(dp), dimension(:, :), allocatable, target onf::pdsds [private]
```

Definition at line 93 of file [onf.f90](#).

Referenced by [inpd\(\)](#), and [readpd\(\)](#).

5.13.4.16 pfan

```
character(len=14), parameter, public onf::pfan = 'Parameter File'
```

Definition at line 86 of file [onf.f90](#).

Referenced by [innf\(\)](#).

5.13.4.17 plfan

```
character(len=14), parameter, public onf::plfan = 'Plankton File'
```

Definition at line 86 of file [onf.f90](#).

Referenced by [innf\(\)](#).

5.13.4.18 sufan

```
character(len=14), parameter, public onf::sufan = 'Spin-up File'
```

Definition at line 86 of file [onf.f90](#).

Referenced by [innf\(\)](#).

5.13.4.19 timan

```
character(len=14), parameter, public onf::timan = 'Time'
```

Definition at line 86 of file [onf.f90](#).

5.13.4.20 timunt

```
character(len=nf90_max_name), public onf::timunt = 's'
```

Definition at line 90 of file [onf.f90](#).

5.14 plankton Module Reference

Set up and drive marine ecosystem dynamics.

Data Types

- type [bc](#)
Type for boundary conditions. [More...](#)

Functions/Subroutines

- subroutine [aggregate](#) (env, grps, thisbox)
aggregation fluxes, forming detritus
- subroutine [alkalinity](#) (env, thisbox)
- subroutine [boxbc_cd](#) (env, boxes)
Boundary concentrations for modified central differences.
- subroutine [boxbc_upwind](#) (env, boxes)
boundary concentrations for upwind scheme
- subroutine [fpar_vertical](#) (env, boxes, nbox)
- subroutine [lch_chl](#) (ambient, boxes, stv)
light attenuation as a function of Chl
- subroutine [lch_pon](#) (ambient, boxes, stv)
light attenuation as a function of PON
- subroutine, public [plankton_init](#) (control, dfn0, ifn0, ofn0, pfn0, sms0, force, resume)
Read all parameters and boundary-condition information and initialise all modules.
- subroutine, public [plankton_ode](#) (neq, time, y, ydot)
Rates of change for all ODEs.

Variables

- type([bacteria](#)), [dimension](#)(:), allocatable, target [bacpla](#)
vector of bacteria groups
- type([layer](#)), [dimension](#)(:), allocatable [box](#)
- type([detritus](#)), [dimension](#)(:), allocatable, target [detrit](#)
vector of detritus groups
- type([dicsys](#)), save [dics](#)
- type([dom](#)), [dimension](#)(:), allocatable, target [doma](#)
vector of DOM types
- type([local](#)), target, save, public [envir](#)
- type([state](#)), [dimension](#)(:), allocatable [fungrp](#)
vector of functional groups
- type([ode](#)), target, save, public [parode](#)
- type([phycmo](#)), [dimension](#)(:), allocatable, target [phypla](#)
vector of phytoplankton groups
- type([phydat](#)), save [phys](#)
- type([statevars](#)), target, save, public [states](#)
- type([timing](#)), target, save, public [times](#)
- type([zoocfo](#)), [dimension](#)(:), allocatable, target [zoopla](#)
vector of zooplankton groups

5.14.1 Detailed Description

Set up and drive marine ecosystem dynamics.

5.14.2 Data Type Documentation

5.14.2.1 type plankton::bc

Type for boundary conditions.

Definition at line 17 of file [plankton.f90](#).

Class Members

character(len=100)	name	variable name
type(quantity)	value	value of boundary condition

5.14.3 Function/Subroutine Documentation

5.14.3.1 aggregate()

```
subroutine plankton::aggregate (
    class(local), intent(inout) env,
    type(state), dimension(:), intent(in) grps,
    type(layer), intent(inout) thisbox ) [private]
```

aggregation fluxes, forming detritus

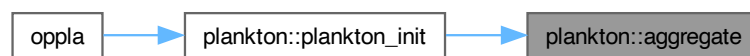
subroutine aggregate cannot be called from the detritus flux routine because the stickiness of functional types might be variable

Definition at line 802 of file [plankton.f90](#).

References [detrit](#).

Referenced by [plankton_init\(\)](#).

Here is the caller graph for this function:



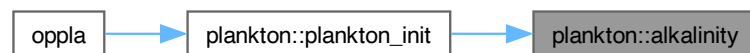
5.14.3.2 alkalinity()

```
subroutine plankton::alkalinity (  
    class(local), intent(in) env,  
    type(layer), intent(inout) thisbox ) [private]
```

Definition at line 818 of file [plankton.f90](#).

Referenced by [plankton_init\(\)](#).

Here is the caller graph for this function:



5.14.3.3 boxbc_cd()

```
subroutine plankton::boxbc_cd (  
    class(local), intent(in) env,  
    type(layer), dimension(:), intent(inout) boxes ) [private]
```

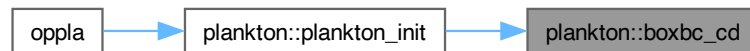
Boundary concentrations for modified central differences.

Where vertical motion follows the gradient, the boundary concentrations are the means of the concentrations in the adjacent layers. Elsewhere they are calculated as the geometric means to prevent negative concentrations.

Definition at line 833 of file [plankton.f90](#).

Referenced by [plankton_init\(\)](#).

Here is the caller graph for this function:



5.14.3.4 boxbc_upwind()

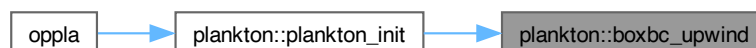
```
subroutine plankton::boxbc_upwind (  
    class(local), intent(in) env,  
    type(layer), dimension(:), intent(inout) boxes ) [private]
```

boundary concentrations for upwind scheme

Definition at line 846 of file [plankton.f90](#).

Referenced by [plankton_init\(\)](#).

Here is the caller graph for this function:



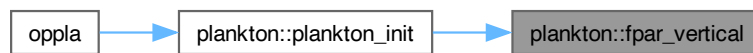
5.14.3.5 fpar_vertical()

```
subroutine plankton::fpar_vertical (  
    class(local), intent(inout) env,  
    type(layer), dimension(:), intent(inout) boxes,  
    integer, intent(in) nbox ) [private]
```

Definition at line 787 of file [plankton.f90](#).

Referenced by [plankton_init\(\)](#).

Here is the caller graph for this function:



5.14.3.6 lch_chl()

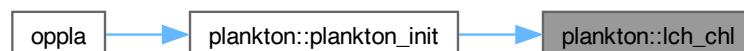
```
subroutine plankton::lch_chl (  
    class(local), intent(in) ambient,  
    type(layer), dimension(:), intent(inout) boxes,  
    real(dp), dimension(:, :), intent(in) stv ) [private]
```

light attenuation as a function of Chl

Definition at line 770 of file [plankton.f90](#).

Referenced by [plankton_init\(\)](#).

Here is the caller graph for this function:



5.14.3.7 lch_pon()

```

subroutine plankton::lch_pon (
    class(local), intent(in) ambient,
    type(layer), dimension(:), intent(inout) boxes,
    real(dp), dimension(:, :), intent(in) stv ) [private]

```

light attenuation as a function of PON

Definition at line 778 of file [plankton.f90](#).

Referenced by [plankton_init\(\)](#).

Here is the caller graph for this function:



5.14.3.8 plankton_init()

```

subroutine, public plankton::plankton_init (
    character(len=*), intent(in) control,
    character(len=*), intent(in) dfn0,
    character(len=*), intent(in) ifn0,
    character(len=*), intent(in) ofn0,
    character(len=*), intent(in) pfn0,
    logical, intent(in) sms0,
    logical, intent(in) force,
    logical, intent(in) resume )

```

Read all parameters and boundary-condition information and initialise all modules.

Parameters

in	<i>control</i>	main input (namelist) file
in	<i>dfn0</i>	NetCDF file with physical forcing (boundary conditions) data
in	<i>ifn0</i>	NetCDF file with initial conditions
in	<i>ofn0</i>	name of output file
in	<i>pfn0</i>	file with parameter namelists
in	<i>resume</i>	flag whether this is a continuation simulation

Parameters

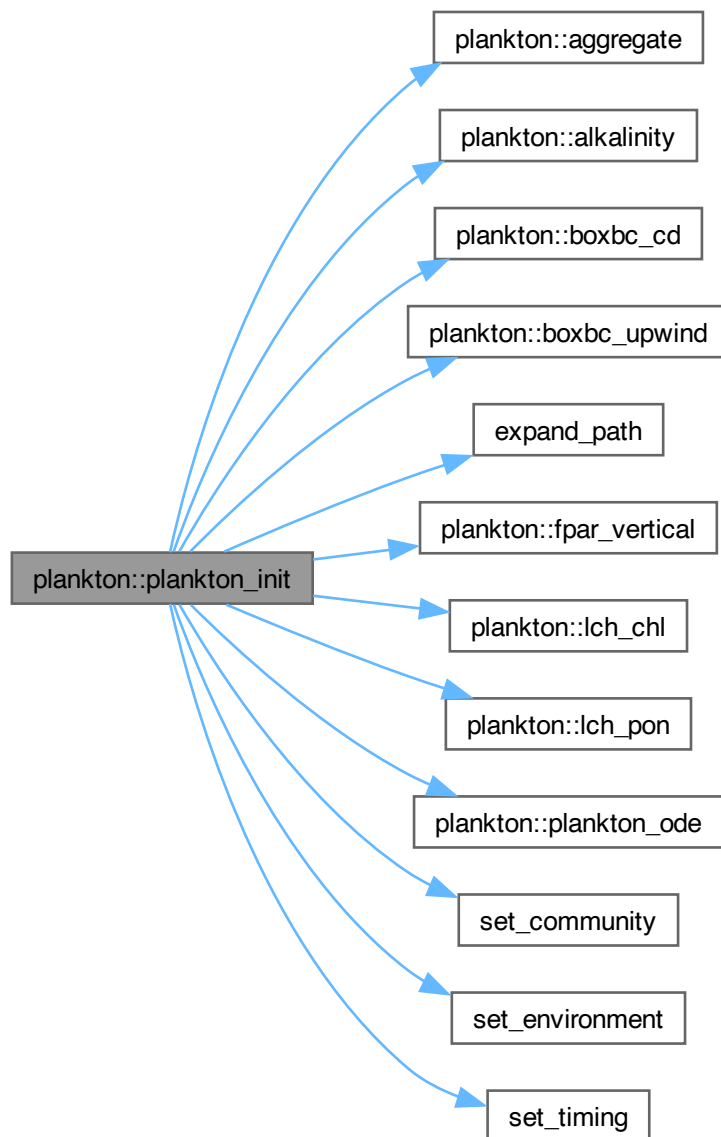
<i>sms0</i>	flag whether to write fluxes to output file
<i>force</i>	flag whether to overwrite output file

Definition at line 42 of file [plankton.f90](#).

References [aggregate\(\)](#), [alkalinity\(\)](#), [box](#), [boxbc_cd\(\)](#), [boxbc_upwind\(\)](#), [et::constituents](#), [detrit](#), [dics](#), [envir](#), [expand_path\(\)](#), [fpar_vertical\(\)](#), [fungrp](#), [lch_chl\(\)](#), [lch_pon\(\)](#), [parode](#), [phypla](#), [phys](#), [plankton_ode\(\)](#), [set_community\(\)](#), [set_environment\(\)](#), [set_timing\(\)](#), [states](#), and [times](#).

Referenced by [oppla\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.14.3.9 plankton_ode()

```

subroutine, public plankton::plankton_ode (
    integer, intent(in) neq,
    real(dp), intent(in) time,
    real(dp), dimension(*), intent(inout), target y,
    real(dp), dimension(*), intent(out), target ydot )
  
```

Rates of change for all ODEs.

This subroutine is passed to VODE_F90 in the main program oppla ([oppla.F90](#)). The state-variable and rates-of-change arrays (y and ydot) are declared as assumed-size (2D) arrays, so that they can be passed as vectors from/to VODE_F90.

Parameters

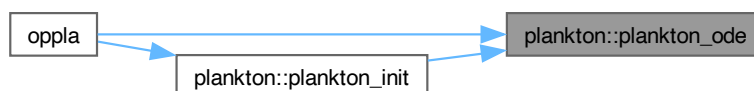
in	<i>neq</i>	number of state variables and ODEs
in	<i>time</i>	current time in s
in, out	<i>y</i>	state variables (stv in oppla and module plankton)
out	<i>ydot</i>	rates of change

Definition at line [862](#) of file [plankton.f90](#).

References [box](#), [dics](#), [envir](#), [fungrp](#), [phys](#), [states](#), and [times](#).

Referenced by [oppla\(\)](#), and [plankton_init\(\)](#).

Here is the caller graph for this function:



5.14.4 Variable Documentation

5.14.4.1 bacpla

```
type(bacteria), dimension(:), allocatable, target plankton::bacpla [private]
```

vector of bacteria groups

Definition at line 27 of file [plankton.f90](#).

Referenced by [set_community\(\)](#).

5.14.4.2 box

```
type(layer), dimension(:), allocatable plankton::box [private]
```

Definition at line 24 of file [plankton.f90](#).

Referenced by [plankton_init\(\)](#), and [plankton_ode\(\)](#).

5.14.4.3 detrit

```
type(detritus), dimension(:), allocatable, target plankton::detrit [private]
```

vector of detritus groups

Definition at line 28 of file [plankton.f90](#).

Referenced by [aggregate\(\)](#), [plankton_init\(\)](#), [set_community\(\)](#), and [set_environment\(\)](#).

5.14.4.4 dics

```
type(dicsys), save plankton::dics [private]
```

Definition at line 31 of file [plankton.f90](#).

Referenced by [plankton_init\(\)](#), [plankton_ode\(\)](#), and [set_environment\(\)](#).

5.14.4.5 doma

```
type(dom), dimension(:), allocatable, target plankton::doma [private]
```

vector of DOM types

Definition at line 26 of file [plankton.f90](#).

Referenced by [set_community\(\)](#).

5.14.4.6 `envir`

```
type(local), target, save, public plankton::envir
```

Definition at line 23 of file [plankton.f90](#).

Referenced by [oppla\(\)](#), [plankton_init\(\)](#), [plankton_ode\(\)](#), [set_community\(\)](#), and [set_environment\(\)](#).

5.14.4.7 `fungrp`

```
type(state), dimension(:), allocatable plankton::fungrp [private]
```

vector of functional groups

Definition at line 25 of file [plankton.f90](#).

Referenced by [plankton_init\(\)](#), [plankton_ode\(\)](#), and [set_community\(\)](#).

5.14.4.8 `parode`

```
type(ode), target, save, public plankton::parode
```

Definition at line 21 of file [plankton.f90](#).

Referenced by [oppla\(\)](#), and [plankton_init\(\)](#).

5.14.4.9 `phypla`

```
type(phycmo), dimension(:), allocatable, target plankton::phypla [private]
```

vector of phytoplankton groups

Definition at line 29 of file [plankton.f90](#).

Referenced by [plankton_init\(\)](#), and [set_community\(\)](#).

5.14.4.10 `phys`

```
type(phydat), save plankton::phys [private]
```

Definition at line 32 of file [plankton.f90](#).

Referenced by [plankton_init\(\)](#), and [plankton_ode\(\)](#).

5.14.4.11 `states`

```
type(statevars), target, save, public plankton::states
```

Definition at line 33 of file [plankton.f90](#).

Referenced by [oppla\(\)](#), [plankton_init\(\)](#), [plankton_ode\(\)](#), [set_community\(\)](#), and [set_environment\(\)](#).

5.14.4.12 times

`type(timing), target, save, public plankton::times`

Definition at line 22 of file [plankton.f90](#).

Referenced by [oppla\(\)](#), [plankton_init\(\)](#), [plankton_ode\(\)](#), [set_environment\(\)](#), and [set_timing\(\)](#).

5.14.4.13 zoopla

`type(zocfo), dimension(:), allocatable, target plankton::zoopla [private]`

vector of zooplankton groups

Definition at line 30 of file [plankton.f90](#).

Referenced by [set_community\(\)](#).

5.15 stdunt Module Reference

common parameters for oppla (dp, stdin, stdout, stderr)

Variables

- integer, parameter `dp = KIND(0D0)`
- integer, parameter `stderr = 0`
- integer, parameter `stdin = 5`
- integer, parameter `stdout = 6`

5.15.1 Detailed Description

common parameters for oppla (dp, stdin, stdout, stderr)

5.15.2 Variable Documentation

5.15.2.1 dp

`integer, parameter stdunt::dp = KIND(0D0)`

Definition at line 4 of file [stdunt.f90](#).

5.15.2.2 stderr

`integer, parameter stdunt::stderr = 0`

Definition at line 4 of file [stdunt.f90](#).

Referenced by [dcode::read_ode\(\)](#).

5.15.2.3 stdin

`integer, parameter stdunt::stdin = 5`

Definition at line 4 of file [stdunt.f90](#).

5.15.2.4 stdout

`integer, parameter stdunt::stdout = 6`

Definition at line 4 of file [stdunt.f90](#).

Chapter 6

Data Type Documentation

6.1 et::aggregation Interface Reference

Calculate aggregation fluxes.

Public Member Functions

- subroutine [aggregation](#) (env, grps, box)

6.1.1 Detailed Description

Calculate aggregation fluxes.

Definition at line [240](#) of file [et.f90](#).

6.1.2 Constructor & Destructor Documentation

6.1.2.1 aggregation()

```
subroutine et::aggregation::aggregation (  
    class(local), intent(inout) env,  
    type(state), dimension(:), intent(in) grps,  
    type(layer), intent(inout) box ) [virtual]
```

Definition at line [240](#) of file [et.f90](#).

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

- [/Users/mpahlow/oppla/src/et.f90](#)

6.2 et::attenuate Interface Reference

Light attenuation.

Public Member Functions

- subroutine [attenuate](#) (ambient, boxes, stv)

6.2.1 Detailed Description

Light attenuation.

Definition at line [255](#) of file [et.f90](#).

6.2.2 Constructor & Destructor Documentation

6.2.2.1 [attenuate\(\)](#)

```
subroutine et::attenuate::attenuate (  
    class(local), intent(in) ambient,  
    type(layer), dimension(:), intent(inout) boxes,  
    real(dp), dimension(:,:), intent(in) stv ) [virtual]
```

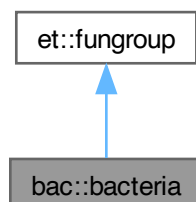
Definition at line [255](#) of file [et.f90](#).

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

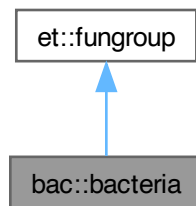
- [/Users/mpahlow/oppla/src/et.f90](#)

6.3 bac::bacteria Type Reference

Inheritance diagram for bac::bacteria:



Collaboration diagram for bac::bacteria:



Public Member Functions

- procedure [flux](#) (grp, grps, env, box, times)
- procedure [read](#) (grp, env, lun)
- procedure [set](#) (grp, grps, env)

Public Member Functions inherited from [et::fungroup](#)

- procedure([fluxes](#)), deferred [flux](#) (grp, grps, env, box, times)
calculate fluxes in the source matrix soma
- procedure [ft_select](#) (grp, [ft](#), caller)
Select and associate temperature-dependence function.
- procedure([init](#)), deferred [read](#) (grp, env, lun)
read parameters of each functional type
- procedure([preset](#)), deferred [set](#) (grp, grps, env)
define ratios, set indices, pointers

Public Attributes

- real([dp](#)) [adim](#) =1.E0_dp
affinity for DIN and DIP
- real([dp](#)) [adop](#) =0._dp
DOP affinity.
- real([dp](#)), pointer [doc](#)
- real([dp](#)), dimension(:), allocatable [docnp](#)
- character(len=100) [dom](#) ='DOM'
name of labile DOM compartment
- real([dp](#)), pointer [don](#)
- real([dp](#)), pointer [dop](#)
- real([dp](#)) [fitem](#)
- real([dp](#)) [ggem](#) =0.3D0
max. bacterial gross growth efficiency
- procedure([bac_grow](#)), pointer [grow](#)
- integer, dimension(:), pointer [idocnp](#)
- integer [idom](#) =0

- `real(dp) lambda = 10._dp`
straightness of the DOC vs. growth relation
- `real(dp) n2p`
N:P ratio.
- `real(dp), pointer qdon`
- `real(dp), pointer qdop`
- `real(dp) qn_param = 0.2`
- `real(dp) qp_param = 0.01`
- `real(dp) rc`
- `procedure(bac_uptake), pointer uptake`
- `real(dp) vdin`
- `real(dp) vdip`
- `real(dp), pointer vdoc`
- `real(dp), dimension(:), allocatable vdom`
- `real(dp), pointer vdon`
- `real(dp), pointer vdop`
- `real(dp) vm`
- `real(dp) vmax = 5D0`
- `real(dp) zeta = 0.6D0`
cost of N assimilation in molN/molC

Public Attributes inherited from `et::fungroup`

- `character(len=100), dimension(:), allocatable constituents`
constituents of this functional group
- `procedure(ft_eppley), pointer ft => ft_Eppley`
- `integer, dimension(:), allocatable icon`
local constituent indices
- `integer ifg`
index of functional group
- `integer, dimension(:), allocatable iq0`
group-sepcific indices of subsistence quotas
- `integer, dimension(:), allocatable ir`
global ratio indices
- `integer, dimension(:), allocatable isv`
vector of state-variable indices
- `logical motile = .FALSE.`
- `character(len=512) name`
name of functional group
- `character(len=100), dimension(:), allocatable names`
names of state variables
- `integer ncon`
number of constituents
- `integer nq0 = 0`
number of subsistence quotas
- `integer nsv = 1`
number of state variables
- `real(dp), pointer oc`
organic C
- `real(dp), dimension(:), allocatable q0`
vector of subsistence quotas

- `real(dp) q10 = 1.89_dp`
- `real(dp)`, pointer `qn`
N:C ratio (cell quota)
- `real(dp)`, pointer `qp`
P:C ratio (cell quota)
- `real(dp)`, `dimension(:)`, pointer `ratios`
variable ratios of this group
- `real(dp)`, pointer `stick`
stickiness ($m^3 (mmol\ C)^{-1}$)
- `real(dp) sticky = 0._dp`
- `real(dp) topt = 15._dp`
- `real(dp) tref = 27._dp`
- `real(dp) tspr = 12._dp`
- `character(len=100)`, `dimension(:)`, allocatable `units`
units of state variables
- `real(dp)`, `dimension(:)`, pointer `vv`
vertical velocities

6.3.1 Detailed Description

Definition at line 7 of file [bac.f90](#).

6.3.2 Member Function/Subroutine Documentation

6.3.2.1 flux()

```
procedure bac::bacteria::flux (
    class(bacteria), intent(inout) grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout) env,
    type(layer), dimension(0:), intent(inout) box,
    type(timing), intent(in) times )
```

Definition at line 28 of file [bac.f90](#).

6.3.2.2 read()

```
procedure bac::bacteria::read (
    class(bacteria), intent(inout), target grp,
    type(local), intent(in), target env,
    integer, intent(in) lun )
```

Definition at line 29 of file [bac.f90](#).

6.3.2.3 set()

```
procedure bac::bacteria::set (
    class(bacteria), intent(inout), target grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout), target env )
```

Definition at line 30 of file [bac.f90](#).

6.3.3 Member Data Documentation

6.3.3.1 adim

```
real(dp) bac::bacteria::adim =1.E0_dp
```

affinity for DIN and DIP

Definition at line 8 of file [bac.f90](#).

6.3.3.2 adop

```
real(dp) bac::bacteria::adop =0._dp
```

DOP affinity.

Definition at line 8 of file [bac.f90](#).

6.3.3.3 doc

```
real(dp), pointer bac::bacteria::doc
```

Definition at line 24 of file [bac.f90](#).

6.3.3.4 docnp

```
real(dp), dimension(:), allocatable bac::bacteria::docnp
```

Definition at line 23 of file [bac.f90](#).

6.3.3.5 dom

```
character(len=100) bac::bacteria::dom ='DOM'
```

name of labile DOM compartment

Definition at line 20 of file [bac.f90](#).

6.3.3.6 don

```
real(dp), pointer bac::bacteria::don
```

Definition at line 24 of file [bac.f90](#).

6.3.3.7 dop

```
real(dp), pointer bac::bacteria::dop
```

Definition at line 24 of file [bac.f90](#).

6.3.3.8 ftem

```
real(dp) bac::bacteria::fitem
```

Definition at line 8 of file [bac.f90](#).

6.3.3.9 ggem

```
real(dp) bac::bacteria::ggem =0.3D0
```

max. bacterial gross growth efficiency

Definition at line 8 of file [bac.f90](#).

6.3.3.10 grow

```
procedure(bac_grow), pointer bac::bacteria::grow
```

Definition at line 25 of file [bac.f90](#).

6.3.3.11 idocnp

```
integer, dimension(:), pointer bac::bacteria::idocnp
```

Definition at line 22 of file [bac.f90](#).

6.3.3.12 idom

```
integer bac::bacteria::idom =0
```

Definition at line 21 of file [bac.f90](#).

6.3.3.13 lambda

```
real(dp) bac::bacteria::lambda =10._dp
```

straightness of the DOC vs. growth relation

Definition at line 8 of file [bac.f90](#).

6.3.3.14 n2p

```
real(dp) bac::bacteria::n2p
```

N:P ratio.

Definition at line 8 of file [bac.f90](#).

6.3.3.15 qdon

```
real(dp), pointer bac::bacteria::qdon
```

Definition at line 24 of file [bac.f90](#).

6.3.3.16 qdop

```
real(dp), pointer bac::bacteria::qdop
```

Definition at line 24 of file [bac.f90](#).

6.3.3.17 qn_param

```
real(dp) bac::bacteria::qn_param =0.2
```

Definition at line 8 of file [bac.f90](#).

6.3.3.18 qp_param

```
real(dp) bac::bacteria::qp_param =0.01
```

Definition at line 8 of file [bac.f90](#).

6.3.3.19 rc

```
real(dp) bac::bacteria::rc
```

Definition at line 8 of file [bac.f90](#).

6.3.3.20 uptake

```
procedure(bac_uptake), pointer bac::bacteria::uptake
```

Definition at line 26 of file [bac.f90](#).

6.3.3.21 vdin

```
real(dp) bac::bacteria::vdin
```

Definition at line 8 of file [bac.f90](#).

6.3.3.22 vdip

```
real(dp) bac::bacteria::vdip
```

Definition at line 8 of file [bac.f90](#).

6.3.3.23 vdoc

```
real(dp), pointer bac::bacteria::vdoc
```

Definition at line 24 of file [bac.f90](#).

6.3.3.24 vdom

```
real(dp), dimension(:), allocatable bac::bacteria::vdom
```

Definition at line 23 of file [bac.f90](#).

6.3.3.25 vdon

```
real(dp), pointer bac::bacteria::vdon
```

Definition at line 24 of file [bac.f90](#).

6.3.3.26 vdop

```
real(dp), pointer bac::bacteria::vdop
```

Definition at line 24 of file [bac.f90](#).

6.3.3.27 vm

```
real(dp) bac::bacteria::vm
```

Definition at line 8 of file [bac.f90](#).

6.3.3.28 vmax

```
real(dp) bac::bacteria::vmax =5D0
```

Definition at line 8 of file [bac.f90](#).

6.3.3.29 zeta

```
real(dp) bac::bacteria::zeta = 0.6D0
```

cost of N assimilation in molN/molC

Definition at line 8 of file [bac.f90](#).

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

- [/Users/mpahlow/oppla/src/bac.f90](#)

6.4 et::boxalk Interface Reference

Public Member Functions

- subroutine [boxalk](#) (env, thisbox)

6.4.1 Detailed Description

Definition at line 275 of file [et.f90](#).

6.4.2 Constructor & Destructor Documentation

6.4.2.1 boxalk()

```
subroutine et::boxalk::boxalk (
    class(local), intent(in) env,
    type(layer), intent(inout) thisbox ) [virtual]
```

Definition at line 275 of file [et.f90](#).

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

- [/Users/mpahlow/oppla/src/et.f90](#)

6.5 et::boxbc Interface Reference

Public Member Functions

- subroutine [boxbc](#) (env, boxes)

6.5.1 Detailed Description

Definition at line 269 of file [et.f90](#).

6.5.2 Constructor & Destructor Documentation

6.5.2.1 boxbc()

```
subroutine et::boxbc::boxbc (
    class(local), intent(in) env,
    type(layer), dimension(:), intent(inout) boxes ) [virtual]
```

Definition at line 269 of file [et.f90](#).

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

- [/Users/mpahlow/oppla/src/et.f90](#)

6.6 julian::c_strftime Interface Reference

Interface to POSIX strftime. Returns a formatted time string, given input time struct and format. See <https://www.cplusplus.com/reference/ctime/strftime> for reference.

Public Member Functions

- integer(c_int) function [c_strftime](#) (str, slen, form, tm)

6.6.1 Detailed Description

Interface to POSIX strftime. Returns a formatted time string, given input time struct and format. See <https://www.cplusplus.com/reference/ctime/strftime> for reference.

Definition at line 81 of file [julian.f90](#).

6.6.2 Constructor & Destructor Documentation

6.6.2.1 c_strftime()

```
integer(c_int) function julian::c_strftime::c_strftime (
    character(kind=c_char), dimension(*), intent(out) str,
    integer(c_int), intent(in), value slen,
    character(kind=c_char), dimension(*), intent(in) form,
    type(tm_struct), intent(in) tm )
```

Definition at line 81 of file [julian.f90](#).

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

- [/Users/mpahlow/oppla/src/julian.f90](#)

6.7 julian::c_strptime Interface Reference

Interface to POSIX strptime. Returns a time struct object based on the input time string *str* and format. See <https://man7.org/linux/man-pages/man3/strptime.3.html> for reference.

Public Member Functions

- `character(kind=c_char, len=1)` function [c_strptime](#) (*str*, *format*, *tm*)

6.7.1 Detailed Description

Interface to POSIX strptime. Returns a time struct object based on the input time string *str* and format. See <https://man7.org/linux/man-pages/man3/strptime.3.html> for reference.

Definition at line 92 of file [julian.f90](#).

6.7.2 Constructor & Destructor Documentation

6.7.2.1 c_strptime()

```
character(kind=c_char, len=1) function julian::c_strptime::c_strptime (  
    character(kind=c_char), dimension(*), intent(in) str,  
    character(kind=c_char), dimension(*), intent(in) format,  
    type(tm_struct), intent(inout) tm )
```

Definition at line 92 of file [julian.f90](#).

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

- `/Users/mpahlow/oppla/src/julian.f90`

6.8 fudu::cv_convert Interface Reference

Public Member Functions

- `real(c_double)` function [cv_convert_double](#) (*cvcnv*, *val*)
- `real(c_float)` function [cv_convert_float](#) (*cvcnv*, *val*)

6.8.1 Detailed Description

Definition at line 80 of file [fudu.F90](#).

6.8.2 Member Function/Subroutine Documentation

6.8.2.1 cv_convert_double()

```
real(c_double) function fudu::cv_convert::cv_convert_double (
    integer(c_intptr_t), value cvcnv,
    real(c_double), value val )
```

Definition at line 87 of file [fudu.F90](#).

6.8.2.2 cv_convert_float()

```
real(c_float) function fudu::cv_convert::cv_convert_float (
    integer(c_intptr_t), value cvcnv,
    real(c_float), value val )
```

Definition at line 81 of file [fudu.F90](#).

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

- [/Users/mpahlow/oppla/src/fudu.F90](#)

6.9 fudu::cv_free Interface Reference

Public Member Functions

- subroutine [cv_free](#) (cvcnv)

6.9.1 Detailed Description

Definition at line 64 of file [fudu.F90](#).

6.9.2 Constructor & Destructor Documentation

6.9.2.1 cv_free()

```
subroutine fudu::cv_free::cv_free (
    integer(c_intptr_t), value cvcnv )
```

Definition at line 64 of file [fudu.F90](#).

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

- [/Users/mpahlow/oppla/src/fudu.F90](#)

6.10 et::decl Interface Reference

daylength function

Public Member Functions

- subroutine [decl](#) (times)

6.10.1 Detailed Description

daylength function

Definition at line [287](#) of file [et.f90](#).

6.10.2 Constructor & Destructor Documentation

6.10.2.1 decl()

```
subroutine et::decl::decl (  
    class(timing), intent(inout) times ) [virtual]
```

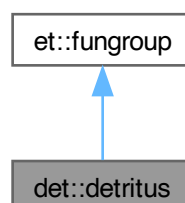
Definition at line [287](#) of file [et.f90](#).

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

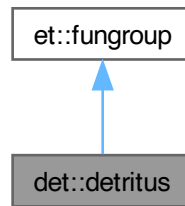
- [/Users/mpahlow/oppla/src/et.f90](#)

6.11 det::detritus Type Reference

Inheritance diagram for det::detritus:



Collaboration diagram for det::detritus:



Public Member Functions

- procedure `flux` (grp, grps, env, box, times)
read detritus-related parameters, set detritus parameters, constituents, number of states
- procedure `read` (grp, env, lun)
flag detritus fluxes in soma for output, set-up remineralisation scheme
- procedure `set` (grp, grps, env)
flag detritus fluxes in soma for output, set-up remineralisation scheme

Public Member Functions inherited from `et::fungroup`

- procedure(`fluxes`), deferred `flux` (grp, grps, env, box, times)
calculate fluxes in the source matrix soma
- procedure `ft_select` (grp, `ft`, caller)
Select and associate temperature-dependence function.
- procedure(`init`), deferred `read` (grp, env, lun)
read parameters of each functional type
- procedure(`preset`), deferred `set` (grp, grps, env)
define ratios, set indices, pointers

Public Attributes

- logical `aggregates` =.FALSE.
flag whether this compartment represents aggregates
- character(len=512) `decay` ="
name of compartment for detritus decay
- logical `detchl`
flag for Chl in detritus
- logical `detpic`
flag for PIC in detritus
- procedure(`det_fluxes`), pointer `fluxes`
- real(`dp`) `formpic` =0._dp
PIC formation rate in s⁻¹.
- integer, dimension(:), allocatable `iloss`
indices of extra loss terms for Chl and O2

- integer, dimension(:), allocatable [irec](#)
indices of recipients of decayed matter
- integer, dimension(:), allocatable [isvl](#)
state-variable indices corresponding to iloss
- real([dp](#)) [kag](#) = -1._dp
aggregation kernel (units should be $m^3 \text{ mmolC}^{-1} \text{ d}^{-1}$)
- real([dp](#)), dimension(:), allocatable [loss](#)
- procedure(det_loss), pointer [losses](#)
- integer [nrec](#)
- real([dp](#)) [omax](#) = 15._dp
maximal Omega above which aragonite will precipitate
- real([dp](#)), dimension(:), allocatable [remi](#)
- real([dp](#)) [remic](#) = 0._dp
- real([dp](#)) [remichl](#) = -1._dp
Chl decay rate in s^{-1} .
- real([dp](#)) [remin](#) = 0._dp
- real([dp](#)) [remip](#) = 0._dp
- real([dp](#)) [remipic](#) = -1._dp
PIC dissolution rate in s^{-1} .
- real([dp](#)) [sink](#) = 0._dp
sinking velocity in $m s^{-1}$

Public Attributes inherited from [et::fungroup](#)

- character(len=100), dimension(:), allocatable [constituents](#)
constituents of this functional group
- procedure([ft_eppley](#)), pointer [ft](#) => [fT_Eppley](#)
- integer, dimension(:), allocatable [icon](#)
local constituent indices
- integer [ifg](#)
index of functional group
- integer, dimension(:), allocatable [iq0](#)
group-sepcific indices of subsistence quotas
- integer, dimension(:), allocatable [ir](#)
global ratio indices
- integer, dimension(:), allocatable [isv](#)
vector of state-variable indices
- logical [motile](#) = .FALSE.
- character(len=512) [name](#)
name of functional group
- character(len=100), dimension(:), allocatable [names](#)
names of state variables
- integer [ncon](#)
number of constituents
- integer [nq0](#) = 0
number of subsistence quotas
- integer [nsv](#) = 1
number of state variables
- real([dp](#)), pointer [oc](#)
organic C
- real([dp](#)), dimension(:), allocatable [q0](#)

- vector of subsistence quotas*
 - real(dp) `q10` =1.89_dp
 - real(dp), pointer `qn`
- N:C ratio (cell quota)*
 - real(dp), pointer `qp`
- P:C ratio (cell quota)*
 - real(dp), dimension(:), pointer `ratios`
- variable ratios of this group*
 - real(dp), pointer `stick`
- stickiness ($m^3 (mmol\ C)^{-1}$)*
 - real(dp) `sticky` =0._dp
 - real(dp) `topt` =15._dp
 - real(dp) `tref` =27._dp
 - real(dp) `tspr` =12._dp
 - character(len=100), dimension(:), allocatable `units`
- units of state variables*
 - real(dp), dimension(:), pointer `vv`
- vertical velocities*

6.11.1 Detailed Description

Definition at line 7 of file [det.f90](#).

6.11.2 Member Function/Subroutine Documentation

6.11.2.1 flux()

```
procedure det::detritus::flux (
    class(detritus), intent(inout) grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout) env,
    type(layer), dimension(0:), intent(inout) box,
    type(timing), intent(in) times )
```

Definition at line 27 of file [det.f90](#).

6.11.2.2 read()

```
procedure det::detritus::read (
    class(detritus), intent(inout), target grp,
    type(local), intent(in), target env,
    integer, intent(in) lun )
```

read detritus-related parameters, set detritus parameters, constituents, number of states

Parameters

in, out	<i>grp</i>	funcional type
in	<i>env</i>	local environment
in	<i>lun</i>	unit number of open file with namelist det

Definition at line 28 of file [det.f90](#).

6.11.2.3 set()

```
procedure det::detritus::set (  
    class(detritus), intent(inout), target grp,  
    type(state), dimension(:), intent(in) grps,  
    type(local), intent(inout), target env )
```

flag detritus fluxes in soma for output, set-up remineralisation scheme

Definition at line 29 of file [det.f90](#).

6.11.3 Member Data Documentation

6.11.3.1 aggregates

```
logical det::detritus::aggregates =.FALSE.
```

flag whether this compartment represents aggregates

Definition at line 17 of file [det.f90](#).

6.11.3.2 decay

```
character(len=512) det::detritus::decay =''
```

name of compartment for detritus decay

Definition at line 16 of file [det.f90](#).

6.11.3.3 detchl

```
logical det::detritus::detchl
```

flag for Chl in detritus

Definition at line 17 of file [det.f90](#).

6.11.3.4 detpic

```
logical det::detritus::detpic
```

flag for PIC in detritus

Definition at line 17 of file [det.f90](#).

6.11.3.5 fluxes

```
procedure(det_fluxes), pointer det::detritus::fluxes
```

Definition at line 24 of file [det.f90](#).

6.11.3.6 formpic

```
real(dp) det::detritus::formpic =0._dp
```

PIC formation rate in s^{-1} .

Definition at line 8 of file [det.f90](#).

6.11.3.7 iloss

```
integer, dimension(:), allocatable det::detritus::iloss
```

indices of extra loss terms for Chl and O₂

Definition at line 21 of file [det.f90](#).

6.11.3.8 irec

```
integer, dimension(:), allocatable det::detritus::irec
```

indices of recipients of decayed matter

Definition at line 21 of file [det.f90](#).

6.11.3.9 isvl

```
integer, dimension(:), allocatable det::detritus::isvl
```

state-variable indices corresponding to iloss

Definition at line 21 of file [det.f90](#).

6.11.3.10 kag

```
real(dp) det::detritus::kag =-1._dp
```

aggregation kernel (units should be $\text{m}^3 \text{mmolC}^{-1} \text{d}^{-1}$)

Definition at line 8 of file [det.f90](#).

6.11.3.11 loss

```
real(dp), dimension(:), allocatable det::detritus::loss
```

Definition at line 15 of file [det.f90](#).

6.11.3.12 losses

```
procedure(det_loss), pointer det::detritus::losses
```

Definition at line 25 of file [det.f90](#).

6.11.3.13 nrec

```
integer det::detritus::nrec
```

Definition at line 20 of file [det.f90](#).

6.11.3.14 omax

```
real(dp) det::detritus::omax =15._dp
```

maximal Omega above which aragonite will precipitate

Definition at line 8 of file [det.f90](#).

6.11.3.15 remi

```
real(dp), dimension(:), allocatable det::detritus::remi
```

Definition at line 15 of file [det.f90](#).

6.11.3.16 remic

```
real(dp) det::detritus::remic =0._dp
```

Definition at line 8 of file [det.f90](#).

6.11.3.17 remichl

```
real(dp) det::detritus::remichl =-1._dp
```

Chl decay rate in s⁻¹.

Definition at line 8 of file [det.f90](#).

6.11.3.18 remin

```
real(dp) det::detritus::remin =0._dp
```

Definition at line 8 of file [det.f90](#).

6.11.3.19 remip

```
real(dp) det::detritus::remip =0._dp
```

Definition at line 8 of file [det.f90](#).

6.11.3.20 remipic

```
real(dp) det::detritus::remipic =-1._dp
```

PIC dissolution rate in s^{-1} .

Definition at line 8 of file [det.f90](#).

6.11.3.21 sink

```
real(dp) det::detritus::sink =0._dp
```

sinking velocity in m s^{-1}

Definition at line 8 of file [det.f90](#).

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

- [/Users/mpahlow/oppla/src/det.f90](#)

6.12 dic::dicsys Type Reference**Public Member Functions**

- procedure [asf](#) (dic, env, box)
Air-to-sea flux of CO_2 and O_2 .
- procedure [co2](#) (dic, box, [niter](#))
Concentrations of protons and DIC species and aragonite and calcite saturation.
- procedure [read](#) (dics, lun, env, [fluxes](#))
Initialise DIC module.

Public Attributes

- `real(dp) ac`
- `real(dp) acd`
- `procedure(dic_alk)`, pointer `alkalinity`
set alkalinity
- `real(dp)`, pointer `asfco2`
- `real(dp)`, pointer `asfo2`
- `real(dp) at`
- `real(dp) bt`
- `real(dp) ca = 10.28E-3_dp`
- `real(dp) ct`
- `real(dp) faco2`
- `real(dp) ft`
- `real(dp) gmk`
- `procedure(hsolve_f06)`, pointer `hsolve`
- `procedure(hini_f06)`, pointer `init`
- `real(dp) k0co2`
- `real(dp) k1`
- `real(dp) k12`
- `real(dp) k12p`
- `real(dp) k1p`
- `real(dp) k2`
- `real(dp) k2p`
- `real(dp) k3p`
- `real(dp) kb`
- `real(dp) kf`
- `real(dp) knh4`
- `real(dp) ksi`
- `real(dp) kso4`
- `real(dp) kspa`
- `real(dp) kspc`
- `real(dp) kw`
- `integer niter = 2`
- `real(dp) ph0 = 8.1_dp`
- `procedure(pivel_lm86)`, pointer `pivel`
calculate transfer velocities for CO₂ and O₂
- `real(dp) potalk = 0._dp`
potential alkalinity (Key et al. 2004) in mmol m⁻³
- `real(dp) pt`
- `real(dp) r_tot_free`
 $[H^+]_T/[H^+]_f$
- `real(dp) r_tot_sws`
 $[H^+]_T/[H^+]_{sws}$
- `real(dp) rrnp`
N:P Redfield ratio.
- `real(dp) rrnsi`
N:Si Redfield ratio.
- `real(dp) schnco2`
- `real(dp) schno2`
- `real(dp) sit`
- `real(dp) so4`
- `real(dp) tk`
- `real(dp) totalk = 0._dp`

- `real(dp) tvco2`
transfer (piston) velocity for CO₂ in m s⁻¹
- `real(dp) tvo2`
transfer (piston) velocity for O₂ in m s⁻¹

6.12.1 Detailed Description

Definition at line 69 of file [dic.f90](#).

6.12.2 Member Function/Subroutine Documentation

6.12.2.1 asf()

```
procedure dic::dicsys::asf (
    class(dicsys), intent(inout) dic,
    type(local), intent(in) env,
    type(layer), intent(inout) box )
```

Air-to-sea flux of CO₂ and O₂.

Oxygen saturation concentration is calculated after [Garcia and Gordon \(1992\)](#), following their recommendation to use the first column of their Table 1.

- See also OCMIP-2

Parameters

in	<code>box%salinity</code>	salinity in psu
in	<code>box%temperature</code>	temperature in °C
in	<code>env%pressure</code>	atmospheric pressure in Pa
in	<code>env%xco2</code>	atmospheric CO ₂ mole fraction
in	<code>env%wind</code>	wind speed at 10 m above surface in m s ⁻¹
in	<code>dic%co2</code>	concentration of CO ₂ in mol kg ⁻¹
out	<code>dic%faco2</code>	fugacity of CO ₂ in air in Pa
out	<code>dic%k0co2</code>	solubility of CO ₂ in mol kg ⁻¹ Pa ⁻¹
out	<code>dic%asfco2</code>	air-sea flux of CO ₂ in mmol m ⁻² s ⁻¹
out	<code>dic%asfo2</code>	air-sea flux of O ₂ in mmol m ⁻² s ⁻¹

Definition at line 92 of file [dic.f90](#).

6.12.2.2 co2()

```
procedure dic::dicsys::co2 (
    class(dicsys), intent(inout) dic,
    type(layer), intent(inout) box,
    integer, intent(in) niter )
```

Concentrations of protons and DIC species and aragonite and calcite saturation.

Parameters

in	<i>box%alkalinity</i>	total alkalinity in $\mu\text{mol L}^{-1}$ (or boxDIN and dicpotalk)
in	<i>box%salinity</i>	salinity in psu
in	<i>box%temperature</i>	temperature in $^{\circ}\text{C}$
in	<i>box%DIC</i>	DIC concentration in mmol m^{-3}
in	<i>box%DIP</i>	phosphate concentration in mmol m^{-3}
in	<i>box%silc</i>	silicate concentration in mmol m^{-3}
in, out	<i>box%hc</i>	proton concentration in mol kg^{-1}
out	<i>box%tk</i>	temperature in K
out	<i>box%CO2</i>	CO_2 concentration in mol kg^{-1}
out	<i>box%HCO3</i>	HCO_3^- concentration in mol kg^{-1}
out	<i>box%CO3</i>	CO_3^{2-} concentration in mol kg^{-1}
out	<i>box%rho</i>	density in g m^{-3}
out	<i>box%omega</i>	aragonite saturation, Millero (1995) , Eq. (81)
out	<i>box%omega_calcite</i>	calcite saturation, Millero (1995) , Eq. (80)

Definition at line 93 of file [dic.f90](#).

6.12.2.3 read()

```

procedure dic::dicsys::read (
    class(dicsys), intent(inout) dics,
    integer, intent(in) lun,
    type(local), intent(inout) env,
    real(dp), dimension(:,0:), intent(in), optional, target fluxes )

```

Initialise DIC module.

pvfun	piston velocity function
LM86	Liss and Merlivat (1986) (default)
W92	Wanninkhof (1992)
WG99	Wanninkhof and McGillis (1999)
W14	Wanninkhof (2014)

Parameters

in	<i>dics%totalk</i>	total alkalinity in mmol m^{-3}
out	<i>dics%rrnsi</i>	N:Si Redfield ratio
out	<i>dics%rrnp</i>	N:P Redfield ratio
out	<i>dics%hc</i>	initial guess for H^+ concentration
in	<i>lun</i>	of file with namelist dic

Definition at line 91 of file [dic.f90](#).

6.12.3 Member Data Documentation

6.12.3.1 ac

```
real(dp) dic::dicsys::ac
```

Definition at line 70 of file [dic.f90](#).

6.12.3.2 acd

```
real(dp) dic::dicsys::acd
```

Definition at line 70 of file [dic.f90](#).

6.12.3.3 alkalinity

```
procedure(dic_alk), pointer dic::dicsys::alkalinity
```

set alkalinity

Definition at line 86 of file [dic.f90](#).

6.12.3.4 asfco2

```
real(dp), pointer dic::dicsys::asfco2
```

Definition at line 84 of file [dic.f90](#).

6.12.3.5 asfo2

```
real(dp), pointer dic::dicsys::asfo2
```

Definition at line 84 of file [dic.f90](#).

6.12.3.6 at

```
real(dp) dic::dicsys::at
```

Definition at line 70 of file [dic.f90](#).

6.12.3.7 bt

```
real(dp) dic::dicsys::bt
```

Definition at line 70 of file [dic.f90](#).

6.12.3.8 ca

```
real(dp) dic::dicsys::ca =10.28E-3_dp
```

Definition at line 70 of file [dic.f90](#).

6.12.3.9 ct

```
real(dp) dic::dicsys::ct
```

Definition at line 70 of file [dic.f90](#).

6.12.3.10 faco2

```
real(dp) dic::dicsys::faco2
```

Definition at line 70 of file [dic.f90](#).

6.12.3.11 ft

```
real(dp) dic::dicsys::ft
```

Definition at line 70 of file [dic.f90](#).

6.12.3.12 gmk

```
real(dp) dic::dicsys::gmk
```

Definition at line 70 of file [dic.f90](#).

6.12.3.13 hsolve

```
procedure(hsolve_f06), pointer dic::dicsys::hsolve
```

Definition at line 88 of file [dic.f90](#).

6.12.3.14 init

```
procedure(hini_f06), pointer dic::dicsys::init
```

Definition at line 89 of file [dic.f90](#).

6.12.3.15 k0co2

```
real(dp) dic::dicsys::k0co2
```

Definition at line 70 of file [dic.f90](#).

6.12.3.16 k1

```
real(dp) dic::dicsys::k1
```

Definition at line 70 of file [dic.f90](#).

6.12.3.17 k12

```
real(dp) dic::dicsys::k12
```

Definition at line 70 of file [dic.f90](#).

6.12.3.18 k12p

```
real(dp) dic::dicsys::k12p
```

Definition at line 70 of file [dic.f90](#).

6.12.3.19 k1p

```
real(dp) dic::dicsys::k1p
```

Definition at line 70 of file [dic.f90](#).

6.12.3.20 k2

```
real(dp) dic::dicsys::k2
```

Definition at line 70 of file [dic.f90](#).

6.12.3.21 k2p

```
real(dp) dic::dicsys::k2p
```

Definition at line 70 of file [dic.f90](#).

6.12.3.22 k3p

```
real(dp) dic::dicsys::k3p
```

Definition at line 70 of file [dic.f90](#).

6.12.3.23 kb

```
real(dp) dic::dicsys::kb
```

Definition at line 70 of file [dic.f90](#).

6.12.3.24 kf

```
real(dp) dic::dicsys::kf
```

Definition at line 70 of file [dic.f90](#).

6.12.3.25 knh4

```
real(dp) dic::dicsys::knh4
```

Definition at line 70 of file [dic.f90](#).

6.12.3.26 ksi

```
real(dp) dic::dicsys::ksi
```

Definition at line 70 of file [dic.f90](#).

6.12.3.27 kso4

```
real(dp) dic::dicsys::kso4
```

Definition at line 70 of file [dic.f90](#).

6.12.3.28 kspa

```
real(dp) dic::dicsys::kspa
```

Definition at line 70 of file [dic.f90](#).

6.12.3.29 kspc

```
real(dp) dic::dicsys::kspc
```

Definition at line 70 of file [dic.f90](#).

6.12.3.30 kw

```
real(dp) dic::dicsys::kw
```

Definition at line 70 of file [dic.f90](#).

6.12.3.31 niter

```
integer dic::dicsys::niter =2
```

Definition at line 85 of file [dic.f90](#).

6.12.3.32 ph0

```
real(dp) dic::dicsys::ph0 =8.1_dp
```

Definition at line 70 of file [dic.f90](#).

6.12.3.33 pivel

```
procedure(pivel_lm86), pointer dic::dicsys::pivel
```

calculate transfer velocities for CO₂ and O₂

Definition at line 87 of file [dic.f90](#).

6.12.3.34 potalk

```
real(dp) dic::dicsys::potalk =0._dp
```

potential alkalinity (Key et al. 2004) in mmol m⁻³

Definition at line 70 of file [dic.f90](#).

6.12.3.35 pt

```
real(dp) dic::dicsys::pt
```

Definition at line 70 of file [dic.f90](#).

6.12.3.36 r_tot_free

```
real(dp) dic::dicsys::r_tot_free
```

$[H^+]_T/[H^+]_f$

Definition at line 70 of file [dic.f90](#).

6.12.3.37 r_tot_sws

```
real(dp) dic::dicsys::r_tot_sws
```

$[H^+]_T/[H^+]_{sws}$

Definition at line 70 of file [dic.f90](#).

6.12.3.38 rrnp

```
real(dp) dic::dicsys::rrnp
```

N:P Redfield ratio.

Definition at line 70 of file [dic.f90](#).

6.12.3.39 rrnsi

```
real(dp) dic::dicsys::rrnsi
```

N:Si Redfield ratio.

Definition at line 70 of file [dic.f90](#).

6.12.3.40 schnco2

```
real(dp) dic::dicsys::schnco2
```

Definition at line 70 of file [dic.f90](#).

6.12.3.41 schno2

```
real(dp) dic::dicsys::schno2
```

Definition at line 70 of file [dic.f90](#).

6.12.3.42 sit

```
real(dp) dic::dicsys::sit
```

Definition at line 70 of file [dic.f90](#).

6.12.3.43 so4

```
real(dp) dic::dicsys::so4
```

Definition at line 70 of file [dic.f90](#).

6.12.3.44 tk

```
real(dp) dic::dicsys::tk
```

Definition at line 70 of file [dic.f90](#).

6.12.3.45 totalk

```
real(dp) dic::dicsys::totalk =0._dp
```

Definition at line 70 of file [dic.f90](#).

6.12.3.46 tvco2

```
real(dp) dic::dicsys::tvco2
```

transfer (piston) velocity for CO₂ in m s⁻¹

Definition at line 70 of file [dic.f90](#).

6.12.3.47 tvo2

```
real(dp) dic::dicsys::tvo2
```

transfer (piston) velocity for O₂ in m s⁻¹

Definition at line 70 of file [dic.f90](#).

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

- [/Users/mpahlow/oppla/src/dic.f90](#)

6.13 onf::dimension Type Reference**Public Member Functions**

- procedure [read](#) (dim, ncid, caller, nferr)

Load dimension variable.

Public Attributes

- type([nf90grd](#)) [bounds](#)
- integer [id](#) =0
dimension ID
- character(len=nf90_max_name) [name](#) ="
- character(len=nf90_max_name) [units](#) ="
- real([dp](#)), [dimension](#)(:), allocatable [values](#)
- integer [varid](#)

dimension-variable ID

6.13.1 Detailed Description

Definition at line 32 of file [onf.f90](#).

6.13.2 Member Function/Subroutine Documentation**6.13.2.1 read()**

```
procedure onf::dimension::read (
    class(dimension), intent(inout) dim,
    integer, intent(in) ncid,
    character(len=*), intent(in) caller,
    integer, intent(out), optional nferr )
```

Load dimension variable.

Parameters

in	<i>dim</i>	TYPE(dimension) variable to read
in	<i>ncid</i>	id of the file or group of the dataset
in	<i>caller</i>	name of the calling subroutine or function
in	<i>dim%name</i>	name of the dimension
out	<i>dim%values</i>	allocatable array to hold the data
out	<i>dim%varid</i>	variable id of the dimension variable
out	<i>nferr</i>	NetCDF error code

Definition at line 39 of file [onf.f90](#).

6.13.3 Member Data Documentation

6.13.3.1 bounds

```
type(nf90grd) onf::dimension::bounds
```

Definition at line 37 of file [onf.f90](#).

6.13.3.2 id

```
integer onf::dimension::id =0
```

dimension ID

Definition at line 33 of file [onf.f90](#).

6.13.3.3 name

```
character(len=nf90_max_name) onf::dimension::name =''
```

Definition at line 35 of file [onf.f90](#).

6.13.3.4 units

```
character(len=nf90_max_name) onf::dimension::units =''
```

Definition at line 35 of file [onf.f90](#).

6.13.3.5 values

```
real(dp), dimension(:), allocatable onf::dimension::values
```

Definition at line 36 of file [onf.f90](#).

6.13.3.6 varid

`integer onf::dimension::varid`

dimension-variable ID

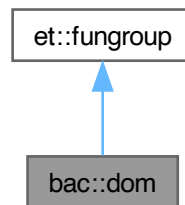
Definition at line 33 of file [onf.f90](#).

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

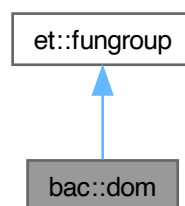
- [/Users/mpahlow/oppla/src/onf.f90](#)

6.14 bac::dom Type Reference

Inheritance diagram for bac::dom:



Collaboration diagram for bac::dom:



Public Member Functions

- procedure [flux](#) (grp, grps, env, box, times)
set DOM-related fluxes in the source matrix soma
- procedure [read](#) (grp, env, lun)
read namelist dom and set dom parameters, constituents, number of states
- procedure [set](#) (grp, grps, env)
flag DOM-related elements of soma for output

Public Member Functions inherited from [et::fungroup](#)

- procedure([fluxes](#)), deferred [flux](#) (grp, grps, env, box, times)
calculate fluxes in the source matrix soma
- procedure [ft_select](#) (grp, ft, caller)
Select and associate temperature-dependence function.
- procedure([init](#)), deferred [read](#) (grp, env, lun)
read parameters of each functional type
- procedure([preset](#)), deferred [set](#) (grp, grps, env)
define ratios, set indices, pointers

Public Attributes

- real([dp](#)) [ldlc](#) =0D0
light-dependent decay rate of labile DOC in $(d\ W\ m^{-2})^{-1} \rightarrow (s\ W\ m^{-2})^{-1}$
- real([dp](#)) [ldln](#) =0D0
light-dependent decay rate of labile DON in $(d\ W\ m^{-2})^{-1} \rightarrow (s\ W\ m^{-2})^{-1}$
- real([dp](#)) [relac](#)
- real([dp](#)) [relan](#)
- real([dp](#)) [trlc](#) =0D0
transformation rate from refractory to labile DOC in $d^{-1} \rightarrow s^{-1}$
- real([dp](#)) [trln](#) =0D0
transformation rate from refractory to labile DON in $d^{-1} \rightarrow s^{-1}$

Public Attributes inherited from [et::fungroup](#)

- character(len=100), dimension(:), allocatable [constituents](#)
constituents of this functional group
- procedure([ft_eppley](#)), pointer ft => ft_Eppley
- integer, dimension(:), allocatable [icon](#)
local constituent indices
- integer [ifg](#)
index of functional group
- integer, dimension(:), allocatable [iq0](#)
group-sepcific indices of subsistence quotas
- integer, dimension(:), allocatable [ir](#)
global ratio indices
- integer, dimension(:), allocatable [isv](#)
vector of state-variable indices
- logical [motile](#) =.FALSE.
- character(len=512) [name](#)
name of functional group
- character(len=100), dimension(:), allocatable [names](#)
names of state variables
- integer [ncon](#)
number of constituents
- integer [nq0](#) =0
number of subsistence quotas
- integer [nsv](#) =1
number of state variables

- real(dp), pointer oc
organic C
- real(dp), dimension(:), allocatable q0
vector of subsistence quotas
- real(dp) q10 = 1.89_dp
- real(dp), pointer qn
N:C ratio (cell quota)
- real(dp), pointer qp
P:C ratio (cell quota)
- real(dp), dimension(:), pointer ratios
variable ratios of this group
- real(dp), pointer stick
stickiness ($m^3 (mmol\ C)^{-1}$)
- real(dp) sticky = 0._dp
- real(dp) topt = 15._dp
- real(dp) tref = 27._dp
- real(dp) tspr = 12._dp
- character(len=100), dimension(:), allocatable units
units of state variables
- real(dp), dimension(:), pointer vv
vertical velocities

6.14.1 Detailed Description

Definition at line 33 of file [bac.f90](#).

6.14.2 Member Function/Subroutine Documentation

6.14.2.1 flux()

```
procedure bac::dom::flux (
    class(dom), intent(inout) grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout) env,
    type(layer), dimension(0:), intent(inout) box,
    type(timing), intent(in) times )
```

set DOM-related fluxes in the source matrix soma

Definition at line 40 of file [bac.f90](#).

6.14.2.2 read()

```
procedure bac::dom::read (
    class(dom), intent(inout), target grp,
    type(local), intent(in), target env,
    integer, intent(in) lun )
```

read namelist dom and set dom parameters, constituents, number of states

Definition at line 41 of file [bac.f90](#).

6.14.2.3 set()

```
procedure bac::dom::set (
    class(dom), intent(inout), target grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout), target env )
```

flag DOM-related elements of soma for output

Definition at line 42 of file [bac.f90](#).

6.14.3 Member Data Documentation

6.14.3.1 ldlc

```
real(dp) bac::dom::ldlc =0D0
```

light-dependent decay rate of labile DOC in $(dW m^{-2})^{-1} \rightarrow (s W m^{-2})^{-1}$

Definition at line 34 of file [bac.f90](#).

6.14.3.2 ldln

```
real(dp) bac::dom::ldln =0D0
```

light-dependent decay rate of labile DON in $(dW m^{-2})^{-1} \rightarrow (s W m^{-2})^{-1}$

Definition at line 34 of file [bac.f90](#).

6.14.3.3 relac

```
real(dp) bac::dom::relac
```

Definition at line 34 of file [bac.f90](#).

6.14.3.4 relan

```
real(dp) bac::dom::relan
```

Definition at line 34 of file [bac.f90](#).

6.14.3.5 trlc

```
real(dp) bac::dom::trlc =0D0
```

transformation rate from refractory to labile DOC in $d^{-1} \rightarrow s^{-1}$

Definition at line 34 of file [bac.f90](#).

6.14.3.6 trln

```
real(dp) bac::dom::trln = 0D0
```

transformation rate from refractory to labile DON in $d^{-1} \rightarrow s^{-1}$

Definition at line 34 of file [bac.f90](#).

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

- [/Users/mpahlow/oppla/src/bac.f90](#)

6.15 et::fluxes Interface Reference

Calculate fluxes in the source matrix soma.

Public Member Functions

- subroutine [fluxes](#) (grp, grps, env, box, times)

6.15.1 Detailed Description

Calculate fluxes in the source matrix soma.

Definition at line 214 of file [et.f90](#).

6.15.2 Constructor & Destructor Documentation

6.15.2.1 fluxes()

```
subroutine et::fluxes::fluxes (
    class(fungroup), intent(inout) grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout) env,
    type(layer), dimension(:), intent(inout) box,
    type(timing), intent(in) times ) [virtual]
```

Definition at line 214 of file [et.f90](#).

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

- [/Users/mpahlow/oppla/src/et.f90](#)

6.16 et::fpar Interface Reference

Fractions of surface PAR arriving at the box bottoms.

Public Member Functions

- subroutine [fpar](#) (env, boxes, nbox)

6.16.1 Detailed Description

Fractions of surface PAR arriving at the box bottoms.

Definition at line 263 of file [et.f90](#).

6.16.2 Constructor & Destructor Documentation

6.16.2.1 fpar()

```
subroutine et::fpar::fpar (
    class(local), intent(inout) env,
    type(layer), dimension(0:), intent(inout) boxes,
    integer, intent(in) nbox ) [virtual]
```

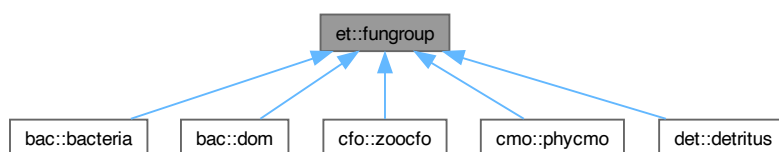
Definition at line 263 of file [et.f90](#).

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

- /Users/mpahlow/oppla/src/[et.f90](#)

6.17 et::fungroup Type Reference

Inheritance diagram for et::fungroup:



Public Member Functions

- procedure([fluxes](#)), deferred [flux](#) (grp, grps, env, box, times)
calculate fluxes in the source matrix soma
- procedure [ft_select](#) (grp, [ft](#), caller)
Select and associate temperature-dependence function.
- procedure([init](#)), deferred [read](#) (grp, env, lun)
read parameters of each functional type
- procedure([preset](#)), deferred [set](#) (grp, grps, env)
define ratios, set indices, pointers

Public Attributes

- character(len=100), dimension(:), allocatable [constituents](#)
constituents of this functional group
- procedure([ft_eppley](#)), pointer [ft](#) => [ft_Eppley](#)
- integer, dimension(:), allocatable [icon](#)
local constituent indices
- integer [ifg](#)
index of functional group
- integer, dimension(:), allocatable [iq0](#)
group-sepcific indices of subsistence quotas
- integer, dimension(:), allocatable [ir](#)
global ratio indices
- integer, dimension(:), allocatable [isv](#)
vector of state-variable indices
- logical [motile](#) =.FALSE.
- character(len=512) [name](#)
name of functional group
- character(len=100), dimension(:), allocatable [names](#)
names of state variables
- integer [ncon](#)
number of constituents
- integer [nq0](#) =0
number of subsistence quotas
- integer [nsv](#) =1
number of state variables
- real([dp](#)), pointer [oc](#)
organic C
- real([dp](#)), dimension(:), allocatable [q0](#)
vector of subsistence quotas
- real([dp](#)) [q10](#) =1.89_dp
- real([dp](#)), pointer [qn](#)
N:C ratio (cell quota)
- real([dp](#)), pointer [qp](#)
P:C ratio (cell quota)
- real([dp](#)), dimension(:), pointer [ratios](#)
variable ratios of this group
- real([dp](#)), pointer [stick](#)
stickiness ($m^3 (mmol\ C)^{-1}$)
- real([dp](#)) [sticky](#) =0._dp
- real([dp](#)) [topt](#) =15._dp
- real([dp](#)) [tref](#) =27._dp
- real([dp](#)) [tspr](#) =12._dp
- character(len=100), dimension(:), allocatable [units](#)
units of state variables
- real([dp](#)), dimension(:), pointer [vv](#)
vertical velocities

6.17.1 Detailed Description

Definition at line 31 of file [et.f90](#).

6.17.2 Member Function/Subroutine Documentation

6.17.2.1 flux()

```
procedure(fluxes), deferred et::fungroup::flux (
    class(fungroup), intent(inout) grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout) env,
    type(layer), dimension(:), intent(inout) box,
    type(timing), intent(in) times ) [pure virtual]
```

calculate fluxes in the source matrix soma

Definition at line 56 of file [et.f90](#).

6.17.2.2 ft_select()

```
procedure et::fungroup::ft_select (
    class(fungroup), intent(inout) grp,
    character(len=*), intent(in) ft,
    character(len=*), intent(in) caller )
```

Select and associate temperature-dependence function.

Parameters

<i>in, out</i>	<i>grp</i>	functional group
<i>in</i>	<i>caller</i>	name of caller (used for error messages)

Parameters

<i>ft</i>	name of temperature function ('oppla', 'Eppley', 'Houlton', or 'ME')
-----------	--

Definition at line 55 of file [et.f90](#).

6.17.2.3 read()

```
procedure(init), deferred et::fungroup::read (
    class(fungroup), intent(inout), target grp,
    type(local), intent(in), target env,
    integer, intent(in) lun ) [pure virtual]
```

read parameters of each functional type

Parameters

<i>in, out</i>	<i>grp</i>	functional type
<i>in</i>	<i>env</i>	local environment
<i>in</i>	<i>lun</i>	logical unit number for namelist input

Definition at line 57 of file [et.f90](#).

6.17.2.4 set()

```
procedure(preset), deferred et::fungroup::set (  
    class(fungroup), intent(inout), target grp,  
    type(state), dimension(:), intent(in) grps,  
    type(local), intent(inout), target env ) [pure virtual]
```

define ratios, set indices, pointers

Definition at line 58 of file [et.f90](#).

6.17.3 Member Data Documentation

6.17.3.1 constituents

```
character(len=100), dimension(:), allocatable et::fungroup::constituents
```

constituents of this functional group

Definition at line 33 of file [et.f90](#).

6.17.3.2 ft

```
procedure(ft_eppley), pointer et::fungroup::ft => fT_Eppley
```

Definition at line 53 of file [et.f90](#).

6.17.3.3 icon

```
integer, dimension(:), allocatable et::fungroup::icon
```

local constituent indices

Definition at line 40 of file [et.f90](#).

6.17.3.4 ifg

```
integer et::fungroup::ifg
```

index of functional group

Definition at line 36 of file [et.f90](#).

6.17.3.5 iq0

```
integer, dimension(:), allocatable et::fungroup::iq0
```

group-sepcific indices of subsistence quotas

Definition at line 40 of file [et.f90](#).

6.17.3.6 ir

```
integer, dimension(:), allocatable et::fungroup::ir
```

global ratio indices

Definition at line 40 of file [et.f90](#).

6.17.3.7 isv

```
integer, dimension(:), allocatable et::fungroup::isv
```

vector of state-variable indices

Definition at line 40 of file [et.f90](#).

6.17.3.8 motile

```
logical et::fungroup::motile =.FALSE.
```

Definition at line 44 of file [et.f90](#).

6.17.3.9 name

```
character(len=512) et::fungroup::name
```

name of functional group

Definition at line 32 of file [et.f90](#).

6.17.3.10 names

```
character(len=100), dimension(:), allocatable et::fungroup::names
```

names of state variables

Definition at line 33 of file [et.f90](#).

6.17.3.11 ncon

```
integer et::fungroup::ncon
```

number of constituents

Definition at line 36 of file [et.f90](#).

6.17.3.12 nq0

```
integer et::fungroup::nq0 =0
```

number of subsistence quotas

Definition at line 36 of file [et.f90](#).

6.17.3.13 nsv

```
integer et::fungroup::nsv =1
```

number of state variables

Definition at line 36 of file [et.f90](#).

6.17.3.14 oc

```
real(dp), pointer et::fungroup::oc
```

organic C

Definition at line 46 of file [et.f90](#).

6.17.3.15 q0

```
real(dp), dimension(:), allocatable et::fungroup::q0
```

vector of subsistence quotas

Definition at line 45 of file [et.f90](#).

6.17.3.16 q10

```
real(dp) et::fungroup::q10 =1.89_dp
```

Definition at line 52 of file [et.f90](#).

6.17.3.17 qn

```
real(dp), pointer et::fungroup::qn
```

N:C ratio (cell quota)

Definition at line 46 of file [et.f90](#).

6.17.3.18 qp

```
real(dp), pointer et::fungroup::qp
```

P:C ratio (cell quota)

Definition at line 46 of file [et.f90](#).

6.17.3.19 ratios

```
real(dp), dimension(:), pointer et::fungroup::ratios
```

variable ratios of this group

Definition at line 46 of file [et.f90](#).

6.17.3.20 stick

```
real(dp), pointer et::fungroup::stick
```

stickiness ($\text{m}^3 (\text{mmol C})^{-1}$)

Definition at line 46 of file [et.f90](#).

6.17.3.21 sticky

```
real(dp) et::fungroup::sticky =0._dp
```

Definition at line 52 of file [et.f90](#).

6.17.3.22 topt

```
real(dp) et::fungroup::topt =15._dp
```

Definition at line 52 of file [et.f90](#).

6.17.3.23 tref

```
real(dp) et::fungroup::tref =27._dp
```

Definition at line 52 of file [et.f90](#).

6.17.3.24 tspr

```
real(dp) et::fungroup::tspr =12._dp
```

Definition at line 52 of file [et.f90](#).

6.17.3.25 units

```
character(len=100), dimension(:), allocatable et::fungroup::units
```

units of state variables

Definition at line 33 of file [et.f90](#).

6.17.3.26 vv

```
real(dp), dimension(:), pointer et::fungroup::vv
```

vertical velocities

Definition at line 46 of file [et.f90](#).

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

- [/Users/mpahlow/oppla/src/et.f90](#)

6.18 et::init Interface Reference

Read namelists.

Public Member Functions

- subroutine [init](#) (grp, env, lun)

6.18.1 Detailed Description

Read namelists.

Definition at line 224 of file [et.f90](#).

6.18.2 Constructor & Destructor Documentation

6.18.2.1 init()

```
subroutine et::init::init (
    class(fungroup), intent(inout), target grp,
    type(local), intent(in), target env,
    integer, intent(in) lun ) [virtual]
```

Parameters

<code>in, out</code>	<i>grp</i>	functional type
<code>in</code>	<i>env</i>	local environment
<code>in</code>	<i>lun</i>	logical unit number for namelist input

Definition at line 224 of file [et.f90](#).

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

- [/Users/mpahlow/oppla/src/et.f90](#)

6.19 `lambert::lambertw` Interface Reference

Public Member Functions

- elemental double precision function [wapd](#) (`x`, `nb`, `l`)
- elemental real function [wapr](#) (`x`, `nb`, `l`)

6.19.1 Detailed Description

Definition at line 7 of file [lambert.f90](#).

6.19.2 Member Function/Subroutine Documentation

6.19.2.1 `wapd()`

```
elemental double precision function lambert::lambertw::wapd (
    double precision, intent(in) x,
    integer, intent(in) nb,
    integer, intent(in) l )
```

Definition at line 246 of file [lambert.f90](#).

6.19.2.2 `wapr()`

```
elemental real function lambert::lambertw::wapr (
    real, intent(in) x,
    integer, intent(in) nb,
    integer, intent(in) l )
```

Definition at line 28 of file [lambert.f90](#).

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

- [/Users/mpahlow/oppla/src/lambert.f90](#)

6.20 et::light Interface Reference

Calculate or read surface irradiance.

Public Member Functions

- subroutine [light](#) (ambient, time)

6.20.1 Detailed Description

Calculate or read surface irradiance.

Definition at line [248](#) of file [et.f90](#).

6.20.2 Constructor & Destructor Documentation

6.20.2.1 light()

```
subroutine et::light::light (  
    class(local), intent(inout) ambient,  
    type(timing), intent(inout) time ) [virtual]
```

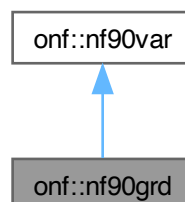
Definition at line [248](#) of file [et.f90](#).

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

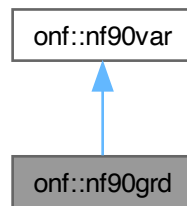
- [/Users/mpahlow/oppla/src/et.f90](#)

6.21 onf::nf90grd Type Reference

Inheritance diagram for onf::nf90grd:



Collaboration diagram for `onf::nf90grd`:



Public Attributes

- `real(kind(0d0))`, `dimension(:, :)`, pointer `values`

Public Attributes inherited from `onf::nf90var`

- `character(len=nf90_max_name)` `name` = "
- integer `ndims`
rank (number of dimensions)
- `character(len=nf90_max_name)` `type` = "
data type, e.g., Temperature, Salinity, etc.
- `character(len=nf90_max_name)` `units` = "
- integer `varid`
variable ID

Additional Inherited Members

Public Member Functions inherited from `onf::nf90var`

- procedure `read` (`ds`, `group`, `data`)
read dataset in group described by ds into array data

6.21.1 Detailed Description

Definition at line 26 of file `onf.f90`.

6.21.2 Member Data Documentation

6.21.2.1 values

```
real(kind(0d0)), dimension(:, :), pointer onf::nf90grd::values
```

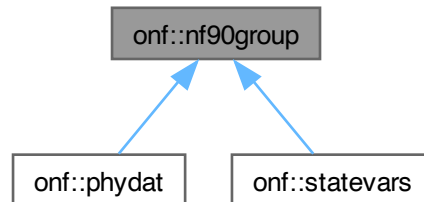
Definition at line 27 of file `onf.f90`.

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

- `/Users/mpahlow/oppla/src/onf.f90`

6.22 onf::nf90group Type Reference

Inheritance diagram for onf::nf90group:



Public Member Functions

- procedure [info](#) (group, latdeg, londeg, t0, t1, writeable)
open groupfilename, read global attributes and coordinates (depth, lon, lat, time)

Public Attributes

- type([nf90stratt](#)), [dimension](#)(:), allocatable [attributes](#)
- type([dimension](#)) [bounds](#)
- type([dimension](#)) [depth](#)
- type([dimension](#)) [depth_flux](#)
- character(len=nf90_max_name) [filename](#)
- integer [id](#) =0
- integer [ilat](#)
- integer [ilon](#)
- integer [it0](#)
- type([dimension](#)) [lat](#)
- type([dimension](#)) [lon](#)
- integer [nbox](#) =0
- integer [ntim](#)
- type([dimension](#)) [state](#)
- type([dimension](#)) [time](#)
- integer [timediff](#) =0

difference between local time and UTC in s

6.22.1 Detailed Description

Definition at line [41](#) of file [onf.f90](#).

6.22.2 Member Function/Subroutine Documentation

6.22.2.1 info()

```
procedure onf::nf90group::info (
    class(nf90group), intent(inout) group,
    real(dp), intent(inout), optional latdeg,
    real(dp), intent(inout), optional londeg,
    real(dp), intent(inout), optional t0,
    real(dp), intent(inout), optional t1,
    logical, intent(in), optional writeable )
```

open groupfilename, read global attributes and coordinates (depth, lon, lat, time)

Parameters

in	<i>group%depth%name</i>	name of depth dataset
in	<i>group%filename</i>	name of file to read
in	<i>group%lat%name</i>	name of latitude dataset
in	<i>group%lon%name</i>	name of longitude dataset
in	<i>group%time%name</i>	name of time dataset
in	<i>group%timediff</i>	difference between local solar time and UTC in s
out	<i>group%depth%values</i>	depth (z) axis
out	<i>group%lat%values</i>	latitude (y) axis
out	<i>group%depth%values</i>	longitude (x) axis
out	<i>group%time%values</i>	time axis
in, out	<i>t0</i>	start time in s
in, out	<i>t1</i>	end time in s
in	<i>writeable</i>	whether to open the file with read-write access

Definition at line 48 of file [onf.f90](#).

6.22.3 Member Data Documentation

6.22.3.1 attributes

```
type(nf90stratt), dimension(:), allocatable onf::nf90group::attributes
```

Definition at line 46 of file [onf.f90](#).

6.22.3.2 bounds

```
type(dimension) onf::nf90group::bounds
```

Definition at line 45 of file [onf.f90](#).

6.22.3.3 depth

```
type(dimension) onf::nf90group::depth
```

Definition at line 45 of file [onf.f90](#).

6.22.3.4 depth_flux

```
type(dimension) onf::nf90group::depth_flux
```

Definition at line 45 of file [onf.f90](#).

6.22.3.5 filename

```
character(len=nf90_max_name) onf::nf90group::filename
```

Definition at line 42 of file [onf.f90](#).

6.22.3.6 id

```
integer onf::nf90group::id =0
```

Definition at line 43 of file [onf.f90](#).

6.22.3.7 ilat

```
integer onf::nf90group::ilat
```

Definition at line 43 of file [onf.f90](#).

6.22.3.8 ilon

```
integer onf::nf90group::ilon
```

Definition at line 43 of file [onf.f90](#).

6.22.3.9 it0

```
integer onf::nf90group::it0
```

Definition at line 43 of file [onf.f90](#).

6.22.3.10 lat

```
type(dimension) onf::nf90group::lat
```

Definition at line 45 of file [onf.f90](#).

6.22.3.11 lon

```
type(dimension) onf::nf90group::lon
```

Definition at line 45 of file [onf.f90](#).

6.22.3.12 nbox

```
integer onf::nf90group::nbox =0
```

Definition at line 43 of file [onf.f90](#).

6.22.3.13 ntim

```
integer onf::nf90group::ntim
```

Definition at line 43 of file [onf.f90](#).

6.22.3.14 state

```
type(dimension) onf::nf90group::state
```

Definition at line 45 of file [onf.f90](#).

6.22.3.15 time

```
type(dimension) onf::nf90group::time
```

Definition at line 45 of file [onf.f90](#).

6.22.3.16 timediff

```
integer onf::nf90group::timediff =0
```

difference between local time and UTC in s

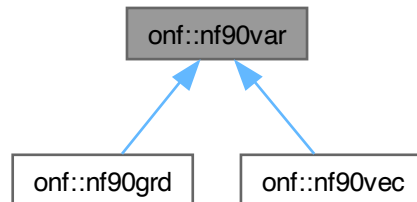
Definition at line 43 of file [onf.f90](#).

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

- [/Users/mpahlow/oppla/src/onf.f90](#)

6.23 onf::nf90var Type Reference

Inheritance diagram for onf::nf90var:



Public Member Functions

- procedure [read](#) (ds, group, data)
read dataset in group described by ds into array data

Public Attributes

- character(len=nf90_max_name) [name](#) ="
- integer [ndims](#)
rank (number of dimensions)
- character(len=nf90_max_name) [type](#) ="
data type, e.g., Temperature, Salinity, etc.
- character(len=nf90_max_name) [units](#) ="
- integer [varid](#)
variable ID

6.23.1 Detailed Description

Definition at line 17 of file [onf.f90](#).

6.23.2 Member Function/Subroutine Documentation

6.23.2.1 read()

```

procedure onf::nf90var::read (
    class(nf90var), intent(inout) ds,
    class(nf90group), intent(in) group,
    real(dp), dimension(*), intent(out) data )
  
```

read dataset in group described by ds into array data

- if dsunits is set, data will be converted to these units

Definition at line 24 of file [onf.f90](#).

6.23.3 Member Data Documentation

6.23.3.1 name

```
character(len=nf90_max_name) onf::nf90var::name =''
```

Definition at line 18 of file [onf.f90](#).

6.23.3.2 ndims

```
integer onf::nf90var::ndims
```

rank (number of dimensions)

Definition at line 21 of file [onf.f90](#).

6.23.3.3 type

```
character(len=nf90_max_name) onf::nf90var::type =''
```

data type, e.g., Temperature, Salinity, etc.

Definition at line 18 of file [onf.f90](#).

6.23.3.4 units

```
character(len=nf90_max_name) onf::nf90var::units =''
```

Definition at line 18 of file [onf.f90](#).

6.23.3.5 varid

```
integer onf::nf90var::varid
```

variable ID

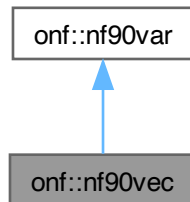
Definition at line 21 of file [onf.f90](#).

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

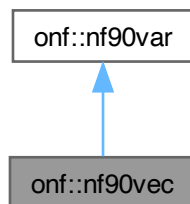
- [/Users/mpahlow/oppla/src/onf.f90](#)

6.24 onf::nf90vec Type Reference

Inheritance diagram for onf::nf90vec:



Collaboration diagram for onf::nf90vec:



Public Attributes

- `real(kind(0d0))`, `dimension(:)`, pointer `values`

Public Attributes inherited from onf::nf90var

- `character(len=nf90_max_name)` `name` = "
- integer `ndims`
rank (number of dimensions)
- `character(len=nf90_max_name)` `type` = "
data type, e.g., Temperature, Salinity, etc.
- `character(len=nf90_max_name)` `units` = "
- integer `varid`
variable ID

Additional Inherited Members

Public Member Functions inherited from `onf::nf90var`

- procedure `read` (ds, group, data)
read dataset in group described by ds into array data

6.24.1 Detailed Description

Definition at line 29 of file `onf.f90`.

6.24.2 Member Data Documentation

6.24.2.1 values

```
real(kind(0d0)), dimension(:), pointer onf::nf90vec::values
```

Definition at line 30 of file `onf.f90`.

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

- `/Users/mpahlow/oppla/src/onf.f90`

6.25 `dvode::ode` Type Reference

Public Member Functions

- procedure `read` (parode, lun, `nconstr`)
Read DVODE parameters.
- procedure `set_atol` (parode, stv)
Set absolute tolerances for each state individually.

Public Attributes

- `real(dp)`, `dimension(:)`, allocatable `atol`
absolute tolerances
- `real(dp)`, `dimension(:)`, allocatable `clower`
lower constraints
- logical `constr` = .TRUE.
- integer, `dimension(:)`, allocatable `constrained`
- `real(dp)`, `dimension(:)`, allocatable `cupper`
upper constraints
- `real(dp)` `h0` = 0D0
- `real(dp)` `hmax` = 0D0
- `real(dp)` `hmin` = 0D0
- integer `itask` = 4
- integer `jsv` = 1

- integer `maxord` =12
- integer `meth` =1
- integer `mf`
- integer `miter` =3
- integer `moss` =0
- real(`dp`), dimension(:), allocatable `mxatol`
maxima of atol
- integer `mxhnil` =0
- real(`dp`), dimension(:), allocatable `mxrtol`
maxima of rtol
- integer `mxstep` =500
- character(len=nf90_max_name), dimension(:), allocatable `names`
- integer `nconstr` =0
- integer `neq` =0
- logical `quiet` =.FALSE.
- real(`dp`), dimension(:), allocatable `rtol`
relative tolerances
- logical `sparse_jacobian`

6.25.1 Detailed Description

Definition at line 9 of file `dcode.f90`.

6.25.2 Member Function/Subroutine Documentation

6.25.2.1 read()

```
procedure dcode::ode::read (
    class(ode), intent(inout) parode,
    integer, intent(in) lun,
    integer, intent(in) nconstr )
```

Read DVODE parameters.

Parameters

in	<code>lun</code>	logical unit number for namelist vode
in	<code>nconstr</code>	number of constrained states

Definition at line 23 of file `dcode.f90`.

6.25.2.2 set_atol()

```
procedure dcode::ode::set_atol (
    class(ode), intent(inout) parode,
    real(dp), dimension(*), intent(in) stv )
```

Set absolute tolerances for each state individually.

Absolute tolerances atol and mxatol are assigned individually if provided as negative values in namelist vode.

Definition at line 24 of file `dcode.f90`.

6.25.3 Member Data Documentation

6.25.3.1 atol

```
real(dp), dimension(:), allocatable dvode::ode::atol
```

absolute tolerances

Definition at line 10 of file [dvode.f90](#).

6.25.3.2 clower

```
real(dp), dimension(:), allocatable dvode::ode::clower
```

lower constraints

Definition at line 10 of file [dvode.f90](#).

6.25.3.3 constr

```
logical dvode::ode::constr =.TRUE.
```

Definition at line 21 of file [dvode.f90](#).

6.25.3.4 constrained

```
integer, dimension(:), allocatable dvode::ode::constrained
```

Definition at line 19 of file [dvode.f90](#).

6.25.3.5 cupper

```
real(dp), dimension(:), allocatable dvode::ode::cupper
```

upper constraints

Definition at line 10 of file [dvode.f90](#).

6.25.3.6 h0

```
real(dp) dvode::ode::h0 =0D0
```

Definition at line 16 of file [dvode.f90](#).

6.25.3.7 hmax

```
real(dp) dvode::ode::hmax =0D0
```

Definition at line 16 of file [dvode.f90](#).

6.25.3.8 hmin

```
real(dp) dvode::ode::hmin =0D0
```

Definition at line 16 of file [dvode.f90](#).

6.25.3.9 itask

```
integer dvode::ode::itask =4
```

Definition at line 17 of file [dvode.f90](#).

6.25.3.10 jsv

```
integer dvode::ode::jsv =1
```

Definition at line 17 of file [dvode.f90](#).

6.25.3.11 maxord

```
integer dvode::ode::maxord =12
```

Definition at line 17 of file [dvode.f90](#).

6.25.3.12 meth

```
integer dvode::ode::meth =1
```

Definition at line 17 of file [dvode.f90](#).

6.25.3.13 mf

```
integer dvode::ode::mf
```

Definition at line 17 of file [dvode.f90](#).

6.25.3.14 miter

```
integer dvode::ode::miter =3
```

Definition at line 17 of file [dvode.f90](#).

6.25.3.15 moss

```
integer dcode::ode::moss =0
```

Definition at line 17 of file [dcode.f90](#).

6.25.3.16 mxatol

```
real(dp), dimension(:), allocatable dcode::ode::mxatol
```

maxima of atol

Definition at line 10 of file [dcode.f90](#).

6.25.3.17 mxhnil

```
integer dcode::ode::mxhnil =0
```

Definition at line 17 of file [dcode.f90](#).

6.25.3.18 mxrtol

```
real(dp), dimension(:), allocatable dcode::ode::mxrtol
```

maxima of rtol

Definition at line 10 of file [dcode.f90](#).

6.25.3.19 mxstep

```
integer dcode::ode::mxstep =500
```

Definition at line 17 of file [dcode.f90](#).

6.25.3.20 names

```
character(len=nf90_max_name), dimension(:), allocatable dcode::ode::names
```

Definition at line 20 of file [dcode.f90](#).

6.25.3.21 nconstr

```
integer dcode::ode::nconstr =0
```

Definition at line 17 of file [dcode.f90](#).

6.25.3.22 neq

```
integer dcode::ode::neq =0
```

Definition at line 17 of file [dcode.f90](#).

6.25.3.23 quiet

```
logical dcode::ode::quiet =.FALSE.
```

Definition at line 21 of file [dcode.f90](#).

6.25.3.24 rtol

```
real(dp), dimension(:), allocatable dcode::ode::rtol
```

relative tolerances

Definition at line 10 of file [dcode.f90](#).

6.25.3.25 sparse_jacobian

```
logical dcode::ode::sparse_jacobian
```

Definition at line 21 of file [dcode.f90](#).

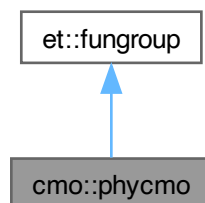
The documentation for this type was generated from the following file:

- [/Users/mpahlow/oppla/src/dcode.f90](#)

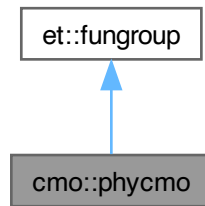
6.26 cmo::phycmo Type Reference

Ordinary phytoplankton and diazotrophs.

Inheritance diagram for cmo::phycmo:



Collaboration diagram for cmo::phycmo:



Public Member Functions

- procedure [flux](#) (grp, grps, env, box, times)
- procedure [read](#) (grp, env, lun)
Read parameters for optimality-based chain model.
- procedure [set](#) (grp, grps, env)

Public Member Functions inherited from [et::fungroup](#)

- procedure([fluxes](#)), deferred [flux](#) (grp, grps, env, box, times)
calculate fluxes in the source matrix soma
- procedure [ft_select](#) (grp, [ft](#), caller)
Select and associate temperature-dependence function.
- procedure([init](#)), deferred [read](#) (grp, env, lun)
read parameters of each functional type
- procedure([preset](#)), deferred [set](#) (grp, grps, env)
define ratios, set indices, pointers

Public Attributes

- real([dp](#)) [a](#)
short form of light-dependent terms
- real([dp](#)) [a0](#)
nutrient affinity in $\text{m}^3 \text{mol}^{-1} \text{d}^{-1}$
- real([dp](#)) [alpha](#)
light affinity in $\text{m}^2 \text{d}^{-1} \text{mol}^{-1} \text{gChl}^{-1}$
- real([dp](#)) [alphat](#)
temperature-dependent alpha
- procedure([cmo_pic](#)), pointer [calc](#)
calcification
- real([dp](#)), pointer [chl](#)
Chl concentration in gChl m^{-3} .
- real([dp](#)) [chl0](#) = -1.0_dp
initial Chl (for Alex' and Steffi's data)

- procedure(chl_dyn), pointer chldyn
light acclimation
- real(dp), pointer daylen
daylength as a fraction of 24h
- real(dp), pointer dlfa
day length factor for A
- character(len=100) dom ='DOM'
name of labile DOM compartment
- logical dynamicchl =.TRUE.
- real(dp) f0n =0._dp
maximum N₂ fixation in molN molC⁻¹ d⁻¹
- real(dp) fc
N allocation to C fixation.
- real(dp) fcdtdq =0._dp
$$\frac{f^C}{\theta} \frac{d\theta}{dQ^N}$$
- real(dp) ff =0._dp
N allocation to N₂ fixation (0 for ordinary phytoplankton, 1 for diazotrophs)
- real(dp) fn
local N allocation to N acquisition
- real(dp) fpic =0._dp
*PIC factor: VPIC (calcification) = fPIC * VDIC (CO₂ fixation)*
- real(dp), pointer ftalpha
temperature factor for alpha
- real(dp) ftemp
temperature factor
- procedure(ftnf_eppley), pointer ftnf => ftnf_Eppley
temperature function for N₂ fixation
- real(dp), pointer ftrc
temperature factor for RC
- real(dp) fv
N allocation to nutrient acquisition.
- procedure(grow), pointer growth
growth function
- integer, dimension(:), allocatable ic
- integer ichl
- integer idoc
- integer ipic
- real(dp) mu0
potential rate of C fixation in d⁻¹
- real(dp) n2f =0._dp
rate of N₂ fixation
- real(dp), pointer ngr
rate of N, P, C acquisition
- procedure(uptake), pointer nuts
nutrient uptake function
- procedure(tchdyn), pointer pachl
chloroplast Chl:C ratio
- real(dp), pointer par
irradiance in mol m⁻² d⁻¹
- logical pic =.FALSE.

- real(dp), pointer q0n
- real(dp), pointer q0p
subsistence N, P quotas
- real(dp), pointer qp_{ic}
PIC:POC ratio.
- real(dp) qsn = 0.02
structure-bound N quota in mol mol⁻¹
- real(dp) rc = 0.1_dp
Chl maintenance respiration in d⁻¹.
- real(dp) rct
temperature-dependent RC
- real(dp) rctht
rate of change of the Chl:C ratio
- real(dp) rdl = 1.0
day-length parameter
- real(dp) si
degree of light saturation
- real(dp) svph
 $\sqrt{\widehat{V}_0^P}$
- real(dp) tch
local (chloroplast) Chl:C ratio in g mol⁻¹
- real(dp), pointer theta
Chl:C ratio in gChl mol⁻¹.
- real(dp) v0 = 5._dp
maximum rate parameter in d⁻¹
- real(dp), dimension(:), allocatable vc
- real(dp), pointer vdic
CO₂ fixation
- real(dp), pointer vdoc
DOC loss (VDOC ≤ 0) under unbalanced growth.
- real(dp) vn
- real(dp) vp
rates of N, P uptake
- real(dp) vph0
local rate of P uptake
- real(dp) vp_{ic}
PIC production (calcification)
- real(dp) vsink = 0._dp
sinking velocity in m d⁻¹
- real(dp) zc
cost of photosynthesis in mol gChl⁻¹
- real(dp) zn
cost of biosynthesis in mol mol⁻¹
- real(dp) znf = 2._dp
cost of biosynthesis for N₂ fixation
- real(dp) znu = 0.6_dp
cost of biosynthesis for DIN use

Public Attributes inherited from `et::fungroup`

- character(len=100), dimension(:), allocatable `constituents`
constituents of this functional group
- procedure(`ft_eppley`), pointer `ft` => `ft_Eppley`
- integer, dimension(:), allocatable `icon`
local constituent indices
- integer `ifg`
index of functional group
- integer, dimension(:), allocatable `iq0`
group-sepcific indices of subsistence quotas
- integer, dimension(:), allocatable `ir`
global ratio indices
- integer, dimension(:), allocatable `isv`
vector of state-variable indices
- logical `motile` = .FALSE.
- character(len=512) `name`
name of functional group
- character(len=100), dimension(:), allocatable `names`
names of state variables
- integer `ncon`
number of constituents
- integer `nq0` = 0
number of subsistence quotas
- integer `nsv` = 1
number of state variables
- real(`dp`), pointer `oc`
organic C
- real(`dp`), dimension(:), allocatable `q0`
vector of subsistence quotas
- real(`dp`) `q10` = 1.89_`dp`
- real(`dp`), pointer `qn`
N:C ratio (cell quota)
- real(`dp`), pointer `qp`
P:C ratio (cell quota)
- real(`dp`), dimension(:), pointer `ratios`
variable ratios of this group
- real(`dp`), pointer `stick`
stickiness ($m^3 (mmol\ C)^{-1}$)
- real(`dp`) `sticky` = 0._`dp`
- real(`dp`) `topt` = 15._`dp`
- real(`dp`) `tref` = 27._`dp`
- real(`dp`) `tspr` = 12._`dp`
- character(len=100), dimension(:), allocatable `units`
units of state variables
- real(`dp`), dimension(:), pointer `vv`
vertical velocities

6.26.1 Detailed Description

Ordinary phytoplankton and diazotrophs.

if $fF = 0$, it is ordinary phytoplankton, if $fF = 1$, it is a diazotroph

Definition at line 13 of file [cmo.f90](#).

6.26.2 Member Function/Subroutine Documentation

6.26.2.1 flux()

```
procedure cmo::phycmo::flux (
    class(phycmo), intent(inout) grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout) env,
    type(layer), dimension(0:), intent(inout) box,
    type(timing), intent(in) times )
```

Definition at line 70 of file [cmo.f90](#).

6.26.2.2 read()

```
procedure cmo::phycmo::read (
    class(phycmo), intent(inout), target grp,
    type(local), intent(in), target env,
    integer, intent(in) lun )
```

Read parameters for optimality-based chain model.

Parameters

<i>in, out</i>	<i>grp</i>	funcional type
<i>in</i>	<i>env</i>	local environment
<i>in</i>	<i>lun</i>	logical unit number for namelist cmo

Definition at line 71 of file [cmo.f90](#).

6.26.2.3 set()

```
procedure cmo::phycmo::set (
    class(phycmo), intent(inout), target grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout), target env )
```

Definition at line 72 of file [cmo.f90](#).

6.26.3 Member Data Documentation

6.26.3.1 a

`real(dp) cmo::phycmo::a`

short form of light-dependent terms

Definition at line 14 of file [cmo.f90](#).

6.26.3.2 a0

`real(dp) cmo::phycmo::a0`

nutrient affinity in $\text{m}^3 \text{mol}^{-1} \text{d}^{-1}$

Definition at line 14 of file [cmo.f90](#).

6.26.3.3 alpha

`real(dp) cmo::phycmo::alpha`

light affinity in $\text{m}^2 \text{d}^{-1} \text{mol W}^{-1} \text{gChl}^{-1}$

Definition at line 14 of file [cmo.f90](#).

6.26.3.4 alphas

`real(dp) cmo::phycmo::alphas`

temperature-dependent alpha

Definition at line 14 of file [cmo.f90](#).

6.26.3.5 calc

`procedure(cmo_pic), pointer cmo::phycmo::calc`

calcification

Definition at line 68 of file [cmo.f90](#).

6.26.3.6 chl

`real(dp), pointer cmo::phycmo::chl`

Chl concentration in gChl m^{-3} .

Definition at line 46 of file [cmo.f90](#).

6.26.3.7 chl0

```
real(dp) cmo::phycmo::chl0 = -1.0_dp
```

initial Chl (for Alex' and Steffi's data)

Definition at line 14 of file [cmo.f90](#).

6.26.3.8 chldyn

```
procedure(chl_dyn), pointer cmo::phycmo::chldyn
```

light acclimation

Definition at line 66 of file [cmo.f90](#).

6.26.3.9 daylen

```
real(dp), pointer cmo::phycmo::daylen
```

daylength as a fraction of 24h

Definition at line 46 of file [cmo.f90](#).

6.26.3.10 dlfa

```
real(dp), pointer cmo::phycmo::dlfa
```

day length factor for A

Definition at line 46 of file [cmo.f90](#).

6.26.3.11 dom

```
character(len=100) cmo::phycmo::dom = 'DOM'
```

name of labile DOM compartment

Definition at line 59 of file [cmo.f90](#).

6.26.3.12 dynamicchl

```
logical cmo::phycmo::dynamicchl = .TRUE.
```

Definition at line 62 of file [cmo.f90](#).

6.26.3.13 f0n

```
real(dp) cmo::phycmo::f0n = 0._dp
```

maximum N₂ fixation in molN molC⁻¹ d⁻¹

Definition at line 14 of file [cmo.f90](#).

6.26.3.14 fc

```
real(dp) cmo::phycmo::fc
```

N allocation to C fixation.

Definition at line 14 of file [cmo.f90](#).

6.26.3.15 fcdtdq

```
real(dp) cmo::phycmo::fcdtdq = 0._dp
```

$$\frac{f^C}{\theta} \frac{d\theta}{dQ^N}$$

Definition at line 14 of file [cmo.f90](#).

6.26.3.16 ff

```
real(dp) cmo::phycmo::ff = 0._dp
```

N allocation to N₂ fixation (0 for ordinary phytoplankton, 1 for diazotrophs)

Definition at line 14 of file [cmo.f90](#).

6.26.3.17 fn

```
real(dp) cmo::phycmo::fn
```

local N allocation to N acquisition

Definition at line 14 of file [cmo.f90](#).

6.26.3.18 fpic

```
real(dp) cmo::phycmo::fpic = 0._dp
```

PIC factor: VPIC (calcification) = fPIC * VDIC (CO₂ fixation)

Definition at line 14 of file [cmo.f90](#).

6.26.3.19 ftalpha

```
real(dp), pointer cmo::phycmo::ftalpha
```

temperature factor for alpha

Definition at line 46 of file [cmo.f90](#).

6.26.3.20 ftemp

```
real(dp) cmo::phycmo::ftemp
```

temperature factor

Definition at line 14 of file [cmo.f90](#).

6.26.3.21 ftmf

```
procedure(ftmf_eppley), pointer cmo::phycmo::ftmf => ftmf_Eppley
```

temperature function for N₂ fixation

Definition at line 63 of file [cmo.f90](#).

6.26.3.22 ftrc

```
real(dp), pointer cmo::phycmo::ftrc
```

temperature factor for RC

Definition at line 46 of file [cmo.f90](#).

6.26.3.23 fv

```
real(dp) cmo::phycmo::fv
```

N allocation to nutrient acquisition.

Definition at line 14 of file [cmo.f90](#).

6.26.3.24 growth

```
procedure(grow), pointer cmo::phycmo::growth
```

growth function

Definition at line 64 of file [cmo.f90](#).

6.26.3.25 ic

```
integer, dimension(:), allocatable cmo::phycmo::ic
```

Definition at line 60 of file [cmo.f90](#).

6.26.3.26 ichl

```
integer cmo::phycmo::ichl
```

Definition at line 61 of file [cmo.f90](#).

6.26.3.27 idoc

```
integer cmo::phycmo::idoc
```

Definition at line 61 of file [cmo.f90](#).

6.26.3.28 ipic

```
integer cmo::phycmo::ipic
```

Definition at line 61 of file [cmo.f90](#).

6.26.3.29 mu0

```
real(dp) cmo::phycmo::mu0
```

potential rate of C fixation in d⁻¹

Definition at line 14 of file [cmo.f90](#).

6.26.3.30 n2f

```
real(dp) cmo::phycmo::n2f = 0._dp
```

rate of N₂ fixation

Definition at line 14 of file [cmo.f90](#).

6.26.3.31 ngr

```
real(dp), pointer cmo::phycmo::ngr
```

rate of N, P, C acquisition

Definition at line 46 of file [cmo.f90](#).

6.26.3.32 nuts

```
procedure(uptake), pointer cmo::phycmo::nuts
```

nutrient uptake function

Definition at line [65](#) of file [cmo.f90](#).

6.26.3.33 pachl

```
procedure(tchdyn), pointer cmo::phycmo::pachl
```

chloroplast Chl:C ratio

Definition at line [67](#) of file [cmo.f90](#).

6.26.3.34 par

```
real(dp), pointer cmo::phycmo::par
```

irradiance in $\text{mol m}^{-2} \text{d}^{-1}$

Definition at line [46](#) of file [cmo.f90](#).

6.26.3.35 pic

```
logical cmo::phycmo::pic = .FALSE.
```

Definition at line [62](#) of file [cmo.f90](#).

6.26.3.36 q0n

```
real(dp), pointer cmo::phycmo::q0n
```

Definition at line [46](#) of file [cmo.f90](#).

6.26.3.37 q0p

```
real(dp), pointer cmo::phycmo::q0p
```

subsistence N, P quotas

Definition at line [46](#) of file [cmo.f90](#).

6.26.3.38 qp

```
real(dp), pointer cmo::phycmo::qp
```

PIC:POC ratio.

Definition at line 46 of file [cmo.f90](#).

6.26.3.39 qsn

```
real(dp) cmo::phycmo::qsn = 0.02
```

structure-bound N quota in mol mol⁻¹

Definition at line 14 of file [cmo.f90](#).

6.26.3.40 rc

```
real(dp) cmo::phycmo::rc = 0.1_dp
```

Chl maintenance respiration in d⁻¹.

Definition at line 14 of file [cmo.f90](#).

6.26.3.41 rct

```
real(dp) cmo::phycmo::rct
```

temperature-dependent RC

Definition at line 14 of file [cmo.f90](#).

6.26.3.42 rctht

```
real(dp) cmo::phycmo::rctht
```

rate of change of the Chl:C ratio

Definition at line 14 of file [cmo.f90](#).

6.26.3.43 rdl

```
real(dp) cmo::phycmo::rdl = 1.0
```

day-length parameter

Definition at line 14 of file [cmo.f90](#).

6.26.3.44 si

```
real(dp) cmo::phycmo::si
```

degree of light saturation

Definition at line 14 of file [cmo.f90](#).

6.26.3.45 svph

```
real(dp) cmo::phycmo::svph
```

$$\sqrt{\widehat{V}_0^P}$$

Definition at line 14 of file [cmo.f90](#).

6.26.3.46 tch

```
real(dp) cmo::phycmo::tch
```

local (chloroplast) Chl:C ratio in g mol⁻¹

Definition at line 14 of file [cmo.f90](#).

6.26.3.47 theta

```
real(dp), pointer cmo::phycmo::theta
```

Chl:C ratio in gChl mol⁻¹.

Definition at line 46 of file [cmo.f90](#).

6.26.3.48 v0

```
real(dp) cmo::phycmo::v0 =5._dp
```

maximum rate parameter in d⁻¹

Definition at line 14 of file [cmo.f90](#).

6.26.3.49 vc

```
real(dp), dimension(:), allocatable cmo::phycmo::vc
```

Definition at line 58 of file [cmo.f90](#).

6.26.3.50 vdic

```
real(dp), pointer cmo::phycmo::vdic
```

CO₂ fixation

Definition at line 46 of file [cmo.f90](#).

6.26.3.51 vdoc

```
real(dp), pointer cmo::phycmo::vdoc
```

DOC loss ($VDOC \leq 0$) under unbalanced growth.

Definition at line 46 of file [cmo.f90](#).

6.26.3.52 vn

```
real(dp) cmo::phycmo::vn
```

Definition at line 14 of file [cmo.f90](#).

6.26.3.53 vp

```
real(dp) cmo::phycmo::vp
```

rates of N, P uptake

Definition at line 14 of file [cmo.f90](#).

6.26.3.54 vph0

```
real(dp) cmo::phycmo::vph0
```

local rate of P uptake

Definition at line 14 of file [cmo.f90](#).

6.26.3.55 vplic

```
real(dp) cmo::phycmo::vplic
```

PIC production (calcification)

Definition at line 14 of file [cmo.f90](#).

6.26.3.56 vsink

```
real(dp) cmo::phycmo::vsink =0._dp
```

sinking velocity in m d^{-1}

Definition at line 14 of file [cmo.f90](#).

6.26.3.57 zc

```
real(dp) cmo::phycmo::zc
```

cost of photosynthesis in mol gChl^{-1}

Definition at line 14 of file [cmo.f90](#).

6.26.3.58 zn

```
real(dp) cmo::phycmo::zn
```

cost of biosynthesis in mol mol^{-1}

Definition at line 14 of file [cmo.f90](#).

6.26.3.59 znf

```
real(dp) cmo::phycmo::znf =2._dp
```

cost of biosynthesis for N_2 fixation

Definition at line 14 of file [cmo.f90](#).

6.26.3.60 znu

```
real(dp) cmo::phycmo::znu =0.6_dp
```

cost of biosynthesis for DIN use

Definition at line 14 of file [cmo.f90](#).

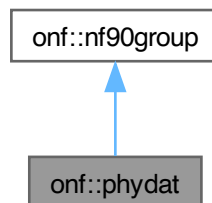
The documentation for this type was generated from the following file:

- [/Users/mpahlow/oppla/src/cmo.f90](#)

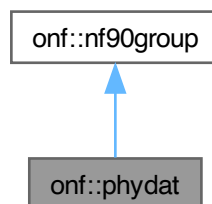
6.27 onf::phydat Type Reference

physical data

Inheritance diagram for onf::phydat:



Collaboration diagram for onf::phydat:



Public Member Functions

- procedure [init](#) (pdf, times, env, box, svd)
Open physical data file and read forcing datasets.

Public Member Functions inherited from [onf::nf90group](#)

- procedure [info](#) (group, latdeg, londeg, t0, t1, writeable)
open groupfilename, read global attributes and coordinates (depth, lon, lat, time)

Public Attributes

- character(len=nf90_max_name) `advect` ='upwind'
- real(`dp`), `dimension`(:), pointer `botbc`
- type(`nf90vec`), `dimension`(:), allocatable `bottom`
- real(`dp`), `dimension`(:,:), allocatable `cds`
- logical `clsbot` =.FALSE.
- type(`nf90grd`), `dimension`(:), allocatable `data`
- real(`dp`) `fpar` =0.43_dp
PAR fraction in total-irradiance data.
- procedure(`rpdp`), pointer `get` => nopd
- integer, `dimension`(:), allocatable `ibot`
- integer, `dimension`(:), allocatable `isflx`
- real(`dp`), `dimension`(:,:), allocatable `pds`
- type(`nf90vec`), `dimension`(:), allocatable `profile`
- real(`dp`), `dimension`(:), allocatable `sds`
- type(`nf90vec`), `dimension`(:), allocatable `surface`
- real(`dp`), `dimension`(:), pointer `surflux`
- type(`nf90vec`), `dimension`(:), allocatable `surflx`
- real(`dp`), `dimension`(:), pointer `surpar`

Public Attributes inherited from `onf::nf90group`

- type(`nf90stratt`), `dimension`(:), allocatable `attributes`
- type(`dimension`) `bounds`
- type(`dimension`) `depth`
- type(`dimension`) `depth_flux`
- character(len=nf90_max_name) `filename`
- integer `id` =0
- integer `ilat`
- integer `ilon`
- integer `it0`
- type(`dimension`) `lat`
- type(`dimension`) `lon`
- integer `nbox` =0
- integer `ntim`
- type(`dimension`) `state`
- type(`dimension`) `time`
- integer `timediff` =0

difference between local time and UTC in s

6.27.1 Detailed Description

physical data

Definition at line 50 of file `onf.f90`.

6.27.2 Member Function/Subroutine Documentation

6.27.2.1 init()

```
procedure onf::phydat::init (
    class(phydat), intent(inout), target pdf,
    type(timing), intent(inout) times,
    type(local), intent(inout) env,
    type(layer), dimension(0:), intent(inout) box,
    type(nf90vec), dimension(:), intent(in) svd )
```

Open physical data file and read forcing datasets.

Subroutine inpd is called as physinit in plankton:plankton_init

Parameters

in, out	<i>pdf%data</i>	gridded (depth-time) forcing data
in, out	<i>pdf%profile</i>	temporally constant vertical profiles
in, out	<i>pdf%surface</i>	surface data

Definition at line 61 of file [onf.f90](#).

6.27.3 Member Data Documentation

6.27.3.1 advect

```
character(len=nf90_max_name) onf::phydat::advect = 'upwind'
```

Definition at line 54 of file [onf.f90](#).

6.27.3.2 botbc

```
real(dp), dimension(:), pointer onf::phydat::botbc
```

Definition at line 55 of file [onf.f90](#).

6.27.3.3 bottom

```
type(nf90vec), dimension(:), allocatable onf::phydat::bottom
```

Definition at line 52 of file [onf.f90](#).

6.27.3.4 cds

```
real(dp), dimension(:, :), allocatable onf::phydat::cds
```

Definition at line 56 of file [onf.f90](#).

6.27.3.5 clsbot

```
logical onf::phydat::clsbot =.FALSE.
```

Definition at line 58 of file [onf.f90](#).

6.27.3.6 data

```
type(nf90grd), dimension(:), allocatable onf::phydat::data
```

Definition at line 53 of file [onf.f90](#).

6.27.3.7 fpar

```
real(dp) onf::phydat::fpar =0.43_dp
```

PAR fraction in total-irradiance data.

Definition at line 57 of file [onf.f90](#).

6.27.3.8 get

```
procedure(rpd), pointer onf::phydat::get => n opd
```

Definition at line 59 of file [onf.f90](#).

6.27.3.9 ibot

```
integer, dimension(:), allocatable onf::phydat::ibot
```

Definition at line 51 of file [onf.f90](#).

6.27.3.10 isflx

```
integer, dimension(:), allocatable onf::phydat::isflx
```

Definition at line 51 of file [onf.f90](#).

6.27.3.11 pds

```
real(dp), dimension(:, :), allocatable onf::phydat::pds
```

Definition at line 56 of file [onf.f90](#).

6.27.3.12 profile

```
type(nf90vec), dimension(:), allocatable onf::phydat::profile
```

Definition at line 52 of file [onf.f90](#).

6.27.3.13 sds

```
real(dp), dimension(:), allocatable onf::phydat::sds
```

Definition at line 56 of file [onf.f90](#).

6.27.3.14 surface

```
type(nf90vec), dimension(:), allocatable onf::phydat::surface
```

Definition at line 52 of file [onf.f90](#).

6.27.3.15 surflux

```
real(dp), dimension(:), pointer onf::phydat::surflux
```

Definition at line 55 of file [onf.f90](#).

6.27.3.16 surflx

```
type(nf90vec), dimension(:), allocatable onf::phydat::surflx
```

Definition at line 52 of file [onf.f90](#).

6.27.3.17 surpar

```
real(dp), dimension(:), pointer onf::phydat::surpar
```

Definition at line 55 of file [onf.f90](#).

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

- [/Users/mpahlow/oppla/src/onf.f90](#)

6.28 et::preset Interface Reference

Define ratios, set indices, pointers.

Public Member Functions

- subroutine [preset](#) (grp, grps, env)

6.28.1 Detailed Description

Define ratios, set indices, pointers.

Definition at line [232](#) of file [et.f90](#).

6.28.2 Constructor & Destructor Documentation

6.28.2.1 preset()

```
subroutine et::preset::preset (
    class(fungroup), intent(inout), target grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout), target env ) [virtual]
```

Definition at line [232](#) of file [et.f90](#).

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

- [/Users/mpahlow/oppla/src/et.f90](#)

6.29 onf::rpd Interface Reference

Public Member Functions

- subroutine [rpd](#) (pdf, time)

6.29.1 Detailed Description

Definition at line [95](#) of file [onf.f90](#).

6.29.2 Constructor & Destructor Documentation

6.29.2.1 rpd()

```
subroutine onf::rpd::rpd (
    class(phydat), intent(inout) pdf,
    real(dp), intent(in) time ) [virtual]
```

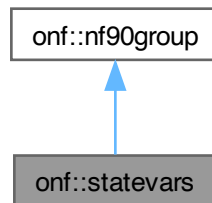
Definition at line [95](#) of file [onf.f90](#).

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

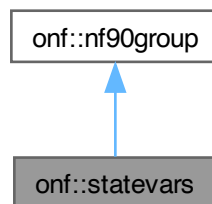
- [/Users/mpahlow/oppla/src/onf.f90](#)

6.30 onf::statevars Type Reference

Inheritance diagram for onf::statevars:



Collaboration diagram for onf::statevars:



Public Member Functions

- procedure [open inmf](#), [init=> invar](#), [write=> outsv](#), [close=> exof](#)

Public Member Functions inherited from [onf::nf90group](#)

- procedure [info](#) (group, latdeg, londeg, t0, t1, writeable)
open groupfilename, read global attributes and coordinates (depth, lon, lat, time)

Public Attributes

- real(dp), [dimension\(:\)](#), allocatable [all_states](#)
- integer, [dimension\(:\)](#), allocatable [count](#)
- real(dp), [dimension\(:, :\)](#), pointer [data](#)
array of state variables (pointer to all_states)
- type(nf90vec) [flux](#)

- real([dp](#)), [dimension](#)(:), allocatable [flux_depth](#)
- integer [i0flux](#)
starting index of fluxes in state-variable vector
- integer [i0sms](#)
starting index of soma in state-variable vector
- integer, [dimension](#)(:), pointer [it](#)
- integer [nfl](#)
No. of flux levels in output.
- integer [nsms](#)
No. of elements in soma to write to output file.
- integer [nsv_all_boxes](#)
total number of plankton state variables (across all layers)
- integer [nsv_one_box](#)
No. of plankton state variables in each layer.
- integer [nsv_total](#)
total number of all (including diagnostic) state variables
- real([dp](#)), [dimension](#)(,:), pointer [roc](#)
array of rates of change (pointer to ydot in plankton_ode)
- logical, [dimension](#)(,:), pointer [smsk](#)
- type([nf90vec](#)) [soma](#)
- integer, [dimension](#)(:), allocatable [start](#)
- character(len=[nf90_max_name](#)) [svgn](#) ="
- type([nf90vec](#)), [dimension](#)(:), allocatable [var](#)
- procedure(write_states), pointer [write_output](#) => write_states
- logical [write_soma](#) =.FALSE.

Public Attributes inherited from [onf::nf90group](#)

- type([nf90stratt](#)), [dimension](#)(:), allocatable [attributes](#)
- type([dimension](#)) [bounds](#)
- type([dimension](#)) [depth](#)
- type([dimension](#)) [depth_flux](#)
- character(len=[nf90_max_name](#)) [filename](#)
- integer [id](#) =0
- integer [ilat](#)
- integer [ilon](#)
- integer [it0](#)
- type([dimension](#)) [lat](#)
- type([dimension](#)) [lon](#)
- integer [nbox](#) =0
- integer [ntim](#)
- type([dimension](#)) [state](#)
- type([dimension](#)) [time](#)
- integer [timediff](#) =0
difference between local time and UTC in s

6.30.1 Detailed Description

Definition at line 63 of file [onf.f90](#).

6.30.2 Member Function/Subroutine Documentation

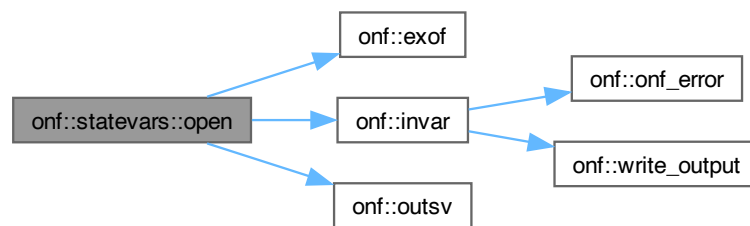
6.30.2.1 open()

procedure onf::statevars::open

Definition at line 83 of file [onf.f90](#).

References [onf::exof\(\)](#), [onf::invar\(\)](#), and [onf::outsv\(\)](#).

Here is the call graph for this function:



6.30.3 Member Data Documentation

6.30.3.1 all_states

real(dp), dimension(:), allocatable onf::statevars::all_states

Definition at line 78 of file [onf.f90](#).

6.30.3.2 count

integer, dimension(:), allocatable onf::statevars::count

Definition at line 72 of file [onf.f90](#).

6.30.3.3 data

real(dp), dimension(:, :), pointer onf::statevars::data

array of state variables (pointer to `all_states`)

Definition at line 76 of file [onf.f90](#).

6.30.3.4 flux

```
type(nf90vec) onf::statevars::flux
```

Definition at line 80 of file [onf.f90](#).

6.30.3.5 flux_depth

```
real(dp), dimension(:), allocatable onf::statevars::flux_depth
```

Definition at line 78 of file [onf.f90](#).

6.30.3.6 i0flux

```
integer onf::statevars::i0flux
```

starting index of fluxes in state-variable vector

Definition at line 64 of file [onf.f90](#).

6.30.3.7 i0sms

```
integer onf::statevars::i0sms
```

starting index of soma in state-variable vector

Definition at line 64 of file [onf.f90](#).

6.30.3.8 it

```
integer, dimension(:), pointer onf::statevars::it
```

Definition at line 71 of file [onf.f90](#).

6.30.3.9 nfl

```
integer onf::statevars::nfl
```

No. of flux levels in output.

Definition at line 64 of file [onf.f90](#).

6.30.3.10 nsms

```
integer onf::statevars::nsms
```

No. of elements in soma to write to output file.

Definition at line 64 of file [onf.f90](#).

6.30.3.11 nsv_all_boxes

```
integer onf::statevars::nsv_all_boxes
```

total number of plankton state variables (across all layers)

Definition at line 64 of file [onf.f90](#).

6.30.3.12 nsv_one_box

```
integer onf::statevars::nsv_one_box
```

No. of plankton state variables in each layer.

Definition at line 64 of file [onf.f90](#).

6.30.3.13 nsv_total

```
integer onf::statevars::nsv_total
```

total number of all (including diagnostic) state variables

Definition at line 64 of file [onf.f90](#).

6.30.3.14 roc

```
real(dp), dimension(:, :), pointer onf::statevars::roc
```

array of rates of change (pointer to ydot in plankton_ode)

Definition at line 76 of file [onf.f90](#).

6.30.3.15 smsk

```
logical, dimension(:, :), pointer onf::statevars::smask
```

Definition at line 74 of file [onf.f90](#).

6.30.3.16 soma

```
type(nf90vec) onf::statevars::soma
```

Definition at line 80 of file [onf.f90](#).

6.30.3.17 start

```
integer, dimension(:), allocatable onf::statevars::start
```

Definition at line 72 of file [onf.f90](#).

6.30.3.18 svgn

```
character(len=nf90_max_name) onf::statevars::svgn = ''
```

Definition at line 75 of file [onf.f90](#).

6.30.3.19 var

```
type(nf90vec), dimension(:), allocatable onf::statevars::var
```

Definition at line 79 of file [onf.f90](#).

6.30.3.20 write_output

```
procedure(write_states), pointer onf::statevars::write_output => write_states
```

Definition at line 81 of file [onf.f90](#).

6.30.3.21 write_soma

```
logical onf::statevars::write_soma =.FALSE.
```

Definition at line 73 of file [onf.f90](#).

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

- [/Users/mpahlow/oppla/src/onf.f90](#)

6.31 et::sunday Interface Reference

Public Member Functions

- subroutine [sunday](#) (times, env)

6.31.1 Detailed Description

Definition at line 281 of file [et.f90](#).

6.31.2 Constructor & Destructor Documentation

6.31.2.1 sunday()

```
subroutine et::sunday::sunday (
    class(timing), intent(inout) times,
    type(local), intent(inout) env ) [virtual]
```

Definition at line 281 of file [et.f90](#).

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

- [/Users/mpahlow/oppla/src/et.f90](#)

6.32 fudu::ut_decode_time Interface Reference

Public Member Functions

- subroutine [ut_decode_time](#) (time, year, month, day, hour, minute, second, res)

6.32.1 Detailed Description

Definition at line 57 of file [fudu.F90](#).

6.32.2 Constructor & Destructor Documentation

6.32.2.1 ut_decode_time()

```
subroutine fudu::ut_decode_time::ut_decode_time (  
    real(c_double), value time,  
    integer(c_int) year,  
    integer(c_int) month,  
    integer(c_int) day,  
    integer(c_int) hour,  
    integer(c_int) minute,  
    real(c_double) second,  
    real(c_double) res )
```

Definition at line 57 of file [fudu.F90](#).

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

- /Users/mpahlow/oppla/src/[fudu.F90](#)

6.33 fudu::ut_encode_time Interface Reference

Public Member Functions

- real(c_double) function [ut_encode_time](#) (year, month, day, hour, minute, second)

6.33.1 Detailed Description

Definition at line 41 of file [fudu.F90](#).

6.33.2 Constructor & Destructor Documentation

6.33.2.1 `ut_encode_time()`

```
real(c_double) function fudu::ut_encode_time::ut_encode_time (
    integer(c_int), value year,
    integer(c_int), value month,
    integer(c_int), value day,
    integer(c_int), value hour,
    integer(c_int), value minute,
    real(c_double), value second )
```

Definition at line 41 of file [fudu.F90](#).

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

- [/Users/mpahlow/oppla/src/fudu.F90](#)

6.34 `fudu::ut_free` Interface Reference

Public Member Functions

- subroutine [ut_free](#) (up)

6.34.1 Detailed Description

Definition at line 69 of file [fudu.F90](#).

6.34.2 Constructor & Destructor Documentation

6.34.2.1 `ut_free()`

```
subroutine fudu::ut_free::ut_free (
    integer(c_intptr_t), value up )
```

Definition at line 69 of file [fudu.F90](#).

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

- [/Users/mpahlow/oppla/src/fudu.F90](#)

6.35 `fudu::ut_free_system` Interface Reference

Public Member Functions

- subroutine [ut_free_system](#) (utsys)

6.35.1 Detailed Description

Definition at line 74 of file [fudu.F90](#).

6.35.2 Constructor & Destructor Documentation

6.35.2.1 ut_free_system()

```
subroutine fudu::ut_free_system::ut_free_system (
    integer(c_intptr_t), value utsys )
```

Definition at line 74 of file [fudu.F90](#).

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

- [/Users/mpahlow/oppla/src/fudu.F90](#)

6.36 fudu::ut_get_converter Interface Reference

Public Member Functions

- integer(c_intptr_t) function [ut_get_converter](#) (up1, up2)

6.36.1 Detailed Description

Definition at line 48 of file [fudu.F90](#).

6.36.2 Constructor & Destructor Documentation

6.36.2.1 ut_get_converter()

```
integer(c_intptr_t) function fudu::ut_get_converter::ut_get_converter (
    integer(c_intptr_t), value up1,
    integer(c_intptr_t), value up2 )
```

Definition at line 48 of file [fudu.F90](#).

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

- [/Users/mpahlow/oppla/src/fudu.F90](#)

6.37 fudu::ut_get_status Interface Reference

Public Member Functions

- integer(c_int) function [ut_get_status](#) ()

6.37.1 Detailed Description

Definition at line 53 of file [fudu.F90](#).

6.37.2 Constructor & Destructor Documentation

6.37.2.1 `ut_get_status()`

```
integer(c_int) function fudu::ut_get_status::ut_get_status
```

Definition at line 53 of file [fudu.F90](#).

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

- [/Users/mpahlow/oppla/src/fudu.F90](#)

6.38 `fudu::ut_parse` Interface Reference

Parse unit string.

Public Member Functions

- `integer(c_intptr_t) function ut_parse (utsys, us, utenc)`

6.38.1 Detailed Description

Parse unit string.

Definition at line 34 of file [fudu.F90](#).

6.38.2 Constructor & Destructor Documentation

6.38.2.1 `ut_parse()`

```
integer(c_intptr_t) function fudu::ut_parse::ut_parse (
    integer(c_intptr_t), value utsys,
    character(kind=c_char) us,
    integer(c_int), value utenc )
```

Parameters

<i>utsys</i>	units system
<i>us</i>	units string
<i>utenc</i>	encoding of units string

Definition at line 34 of file [fudu.F90](#).

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

- [/Users/mpahlow/oppla/src/fudu.F90](#)

6.39 fudu::ut_read_xml Interface Reference

Set-up and return units system from xml database.

Public Member Functions

- `integer(c_intptr_t) function ut_read_xml (file)`

6.39.1 Detailed Description

Set-up and return units system from xml database.

Definition at line 28 of file [fudu.F90](#).

6.39.2 Constructor & Destructor Documentation

6.39.2.1 ut_read_xml()

```
integer(c_intptr_t) function fudu::ut_read_xml::ut_read_xml (
    character(kind=c_char) file )
```

Parameters

<i>file</i>	xml database
-------------	--------------

Definition at line 28 of file [fudu.F90](#).

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

- [/Users/mpahlow/oppla/src/fudu.F90](#)

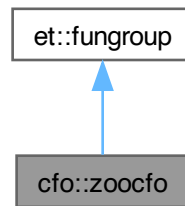
6.40 cfo::zoocfo Type Reference

Zooplankton

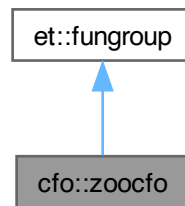
Components depsvm, v0svm, dtsvma, and (doya and doyd) or svm must be set to enable seasonal vertical migration.

Components depdvm and vdvms must be set to enable diel vertical migration.

Inheritance diagram for `cfo::zoocfo`:



Collaboration diagram for `cfo::zoocfo`:



Public Member Functions

- procedure `flux` (grp, grps, env, box, times)
flux routine for the cfo module
- procedure `read` (grp, env, lun)
Read namelist cfo for matching species.
- procedure `set` (grp, grps, env)

Public Member Functions inherited from `et::fungroup`

- procedure(`fluxes`), deferred `flux` (grp, grps, env, box, times)
calculate fluxes in the source matrix soma
- procedure `ft_select` (grp, ft, caller)
Select and associate temperature-dependence function.
- procedure(`init`), deferred `read` (grp, env, lun)
read parameters of each functional type
- procedure(`preset`), deferred `set` (grp, grps, env)
define ratios, set indices, pointers

Public Attributes

- `real(dp) af = 0._dp`
foraging activity
- `real(dp) at`
total activity
- `real(dp) beta = 0.2_dp`
digestion (assimilation) coefficient
- `real(dp) ca = 0.1_dp`
cost of assimilation coefficient
- `real(dp) cf`
cost of foraging coefficient
- `real(dp) cf0 = 0.1_dp`
- `real(dp) cmax = 1._dp`
maximal clearance rate ($\text{m}^3 (\text{mol C})^{-1}$)
- `real(dp)`, pointer `cxda`
product of biomass and doya
- `real(dp)`, pointer `cxdd`
product of biomass and doyd
- `real(dp) d0dvm = 0._dp`
upper depth of diel vertical migration (m)
- `real(dp) d0svm = 0._dp`
upper depth of seasonal vertical migration (m)
- `real(dp) depdvm = 0._dp`
lower depth of diel vertical migration (m)
- `real(dp) depsvm = 0._dp`
lower depth of seasonal vertical migration (m)
- `logical dodvm = .FALSE.`
diel vertical migration
- `logical dodvm0`
default for dodvm
- `character(len=512) dom = 'DOM'`
name of labile DOM compartment
- `logical dosvm = .FALSE.`
seasonal vertical migration
- `real(dp)`, pointer `doya`
day of year of ascent
- `real(dp)`, pointer `doyd`
day of year of descent
- `real(dp) dtdvm = 1.5_dp`
time offset (before sunrise/sunset) for DVM (s)
- `real(dp) dtsvma = 0._dp`
temporal spread around doya (d)
- `real(dp) dtsvmd = 0._dp`
temporal spread around doyd (d)
- `procedure(cfo_dvm)`, pointer `dvmig`
function for diel vertical migration (DVM)
- `logical dynsvm = .FALSE.`
dynamic traits for seasonal vertical migration
- `real(dp) e = 1._dp`
assimilation efficiency

- procedure([egest_pm](#)), pointer [egest](#)
- character(len=512) [egestion](#) = 'det'
name of compartment for egestion
- real([dp](#)) [emax](#) = 0.99_dp
max. assimilation efficiency
- real([dp](#)), dimension(:), allocatable [eq](#)
particulate egestion
- procedure([excrete_dom](#)), pointer [excrete](#)
- real([dp](#)), pointer [fdchl](#)
dissolved fraction of egested Chl
- real([dp](#)), dimension(:), allocatable [fde](#)
dissolved fraction of egestion
- real([dp](#)), pointer [fdpic](#)
dissolved fraction of egested PIC
- real([dp](#)) [fhphi](#) = 1._dp
hibernation factor for phi
- real([dp](#)) [fhrm](#) = 1._dp
hibernation factor for Rm
- procedure([forage_cfo](#)), pointer [forage](#)
- real([dp](#)), dimension(:), allocatable [fpx](#)
particulate fraction of excretion
- real([dp](#)) [fsvm](#)
extension factor of the migration time window
- real([dp](#)) [g](#) = 0._dp
net growth rate
- real([dp](#)) [gmax](#)
max. net growth rate
- real([dp](#)) [i](#) = 0._dp
ingestion
- integer, dimension(:), allocatable [icdet](#)
- integer [id0svm](#) = 0
index of summer-time level for SVM
- integer [id0svm1](#)
id0svm + 1
- integer [id1dvm](#) = 0
index of day-time level for DVM
- integer [id1svm](#) = 0
index of winter-time level for SVM
- integer [id1svm1](#)
id1svm - 1
- integer [ida](#) = 0
- integer [idd](#) = 0
- integer, dimension(:), allocatable [idet](#)
- integer, dimension(:), allocatable [idig](#)
- integer, dimension(:), allocatable [idim](#)
- integer, dimension(:), pointer [idom](#)
- real([dp](#)) [imaf](#)
Imax for ambush feeding.
- real([dp](#)) [imax](#)
max. specific ingestion rate (1/d)
- integer, dimension(:), allocatable [imot](#)

- integer, dimension(:), allocatable [inot](#)
- real([dp](#)), dimension(:), allocatable [ip](#)
individual prey loss rates (prey mortalities)
- integer, pointer [irda](#)
- integer, pointer [irdd](#)
- integer [isda](#)
- integer [isdd](#)
- real([dp](#)) [mort](#) = 0._dp
fish mortality in the surface layer(s)
- real([dp](#)) [ngr](#)
net growth rate
- type([pcc](#)), dimension(:), allocatable [pcc](#)
prey capture coefficient associations
- real([dp](#)), dimension(:), allocatable [phi](#)
food preferences -> prey capture coefficients ($m^3 (mol C)^{-1}$)
- real([dp](#)), dimension(:), allocatable [phi0](#)
default phi for SVM
- real([dp](#)), dimension(:), allocatable [phim](#)
phis for motile prey
- real([dp](#)), dimension(:), allocatable [phin](#)
phis for non-motile prey
- real([dp](#)), dimension(:), allocatable [pp](#)
*effective prey (food concentration*phi)*
- real([dp](#)), dimension(:), allocatable [ppm](#)
same as PP but for motile prey only
- real([dp](#)), dimension(:), allocatable [ppn](#)
same as PP but for non-motile prey only
- real([dp](#)), dimension(:), allocatable [pq](#)
prey cell quotas
- real([dp](#)) [pth](#)
feeding threshold ($mmolC m^{-3}$)
- real([dp](#)) [qn_param](#) = 0.16_dp
- real([dp](#)) [qp_param](#) = 0.01_dp
- real([dp](#)), dimension(:), pointer [quotas](#)
zooplankton nutrient:C ratios
- real([dp](#)) [r](#) = 0._dp
respiration
- real([dp](#)), dimension(:), allocatable [rff](#)
ratio of PP/SUM(PP), corrected for assimilation efficiencies
- real([dp](#)), dimension(:), pointer [rffm](#)
RFF for motile prey.
- real([dp](#)), dimension(:), pointer [rffn](#)
RFF for non-motile prey.
- real([dp](#)) [rm](#)
specific maintenance respiration (1/d)
- real([dp](#)) [rm0](#) = 0._dp
default Rm
- real([dp](#)), dimension(:), allocatable [rq](#)
remineralisation of dissolved inorganic nutrients
- procedure([cfo_svm_static](#)), pointer [svmig](#)
function for seasonal vertical migration (SVM)

- logical `switch` =.FALSE.
switching bewteen motile/non-motile prey
- real(`dp`) `thcsvm` =1.E-6_dp
threshold for svm in mmolC m⁻³
- real(`dp`) `toy`
time of year (decimal days)
- real(`dp`) `v0svm` =0._dp
potential seasonal vertical migration velocity (m/d)
- real(`dp`) `vdvm` =0._dp
diel vertical migration velocity
- real(`dp`), pointer `vsvm`
seasonal vertical migration velocity (m/d)
- real(`dp`) `vsvm0` =0._dp
migration velocity when leaving the surface (m/s)
- real(`dp`) `vsvm1` =0._dp
migration velocity when leaving hibernation depth (m/s)
- real(`dp`), dimension(:), allocatable `xq`
total excretion including egestion

Public Attributes inherited from `et::fungroup`

- character(len=100), dimension(:), allocatable `constituents`
constituents of this functional group
- procedure(`ft_eppley`), pointer `ft` => `ft_Eppley`
- integer, dimension(:), allocatable `icon`
local constituent indices
- integer `ifg`
index of functional group
- integer, dimension(:), allocatable `iq0`
group-sepcific indices of subsistence quotas
- integer, dimension(:), allocatable `ir`
global ratio indices
- integer, dimension(:), allocatable `isv`
vector of state-variable indices
- logical `motile` =.FALSE.
- character(len=512) `name`
name of functional group
- character(len=100), dimension(:), allocatable `names`
names of state variables
- integer `ncon`
number of constituents
- integer `nq0` =0
number of subsistence quotas
- integer `nsv` =1
number of state variables
- real(`dp`), pointer `oc`
organic C
- real(`dp`), dimension(:), allocatable `q0`
vector of subsistence quotas
- real(`dp`) `q10` =1.89_dp

- real(dp), pointer [qn](#)
N:C ratio (cell quota)
- real(dp), pointer [qp](#)
P:C ratio (cell quota)
- real(dp), dimension(:), pointer [ratios](#)
variable ratios of this group
- real(dp), pointer [stick](#)
stickiness ($m^3 (mmol\ C)^{-1}$)
- real(dp) [sticky](#) = 0._dp
- real(dp) [topt](#) = 15._dp
- real(dp) [tref](#) = 27._dp
- real(dp) [tspr](#) = 12._dp
- character(len=100), dimension(:), allocatable [units](#)
units of state variables
- real(dp), dimension(:), pointer [vv](#)
vertical velocities

6.40.1 Detailed Description

Zooplankton

Components `depsvm`, `v0svm`, `dtsvma`, and (`doya` and `doyd`) or `svm` must be set to enable seasonal vertical migration.

Components `depdvm` and `vdvm` must be set to enable diel vertical migration.

Definition at line 21 of file [cfo.f90](#).

6.40.2 Member Function/Subroutine Documentation

6.40.2.1 flux()

```
procedure cfo::zoocfo::flux (
    class(zoocfo), intent(inout) grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout) env,
    type(layer), dimension(0:), intent(inout) box,
    type(timing), intent(in) times )
```

flux routine for the cfo module

Definition at line 106 of file [cfo.f90](#).

6.40.2.2 read()

```
procedure cfo::zoocfo::read (
    class(zoocfo), intent(inout), target grp,
    type(local), intent(in), target env,
    integer, intent(in) lun )
```

Read namelist cfo for matching species.

Definition at line 107 of file [cfo.f90](#).

6.40.2.3 set()

```
procedure cfo::zoocfo::set (
    class(zoocfo), intent(inout), target grp,
    type(state), dimension(:), intent(in) grps,
    type(local), intent(inout), target env )
```

Definition at line 108 of file [cfo.f90](#).

6.40.3 Member Data Documentation

6.40.3.1 af

```
real(dp) cfo::zoocfo::af =0._dp
```

foraging activity

Definition at line 22 of file [cfo.f90](#).

6.40.3.2 at

```
real(dp) cfo::zoocfo::at
```

total activity

Definition at line 22 of file [cfo.f90](#).

6.40.3.3 beta

```
real(dp) cfo::zoocfo::beta =0.2_dp
```

digestion (assimilation) coefficient

Definition at line 22 of file [cfo.f90](#).

6.40.3.4 ca

```
real(dp) cfo::zoocfo::ca =0.1_dp
```

cost of assimilation coefficient

Definition at line 22 of file [cfo.f90](#).

6.40.3.5 cf

```
real(dp) cfo::zoocfo::cf
```

cost of foraging coefficient

Definition at line 22 of file [cfo.f90](#).

6.40.3.6 cf0

```
real(dp) cfo::zoocfo::cf0 =0.1_dp
```

Definition at line 22 of file [cfo.f90](#).

6.40.3.7 cmax

```
real(dp) cfo::zoocfo::cmax =1._dp
```

maximal clearance rate ($\text{m}^3 (\text{mol C})^{-1}$)

Definition at line 22 of file [cfo.f90](#).

6.40.3.8 cxda

```
real(dp), pointer cfo::zoocfo::cxda
```

product of biomass and doya

Definition at line 93 of file [cfo.f90](#).

6.40.3.9 cxdd

```
real(dp), pointer cfo::zoocfo::cxdd
```

product of biomass and doyd

Definition at line 93 of file [cfo.f90](#).

6.40.3.10 d0dvm

```
real(dp) cfo::zoocfo::d0dvm =0._dp
```

upper depth of diel vertical migration (m)

Definition at line 22 of file [cfo.f90](#).

6.40.3.11 d0svm

```
real(dp) cfo::zoocfo::d0svm =0._dp
```

upper depth of seasonal vertical migration (m)

Definition at line 22 of file [cfo.f90](#).

6.40.3.12 depdvm

```
real(dp) cfo::zoocfo::depdvm = 0._dp
```

lower depth of diel vertical migration (m)

Definition at line 22 of file [cfo.f90](#).

6.40.3.13 depsvm

```
real(dp) cfo::zoocfo::depsvm = 0._dp
```

lower depth of seasonal vertical migration (m)

Definition at line 22 of file [cfo.f90](#).

6.40.3.14 dodvm

```
logical cfo::zoocfo::dodvm = .FALSE.
```

diel vertical migration

Definition at line 69 of file [cfo.f90](#).

6.40.3.15 dodvm0

```
logical cfo::zoocfo::dodvm0
```

default for dodvm

Definition at line 69 of file [cfo.f90](#).

6.40.3.16 dom

```
character(len=512) cfo::zoocfo::dom = 'DOM'
```

name of labile DOM compartment

Definition at line 59 of file [cfo.f90](#).

6.40.3.17 dosvm

```
logical cfo::zoocfo::dosvm = .FALSE.
```

seasonal vertical migration

Definition at line 69 of file [cfo.f90](#).

6.40.3.18 doya

```
real(dp), pointer cfo::zoocfo::doya
```

day of year of ascent

Definition at line 93 of file [cfo.f90](#).

6.40.3.19 doyd

```
real(dp), pointer cfo::zoocfo::doyd
```

day of year of descent

Definition at line 93 of file [cfo.f90](#).

6.40.3.20 dtdvm

```
real(dp) cfo::zoocfo::dtdvm =1.5_dp
```

time offset (before sunrise/sunset) for DVM (s)

Definition at line 22 of file [cfo.f90](#).

6.40.3.21 dtsvma

```
real(dp) cfo::zoocfo::dtsvma =0._dp
```

temporal spread around doya (d)

Definition at line 22 of file [cfo.f90](#).

6.40.3.22 dtsvmd

```
real(dp) cfo::zoocfo::dtsvmd =0._dp
```

temporal spread around doyd (d)

Definition at line 22 of file [cfo.f90](#).

6.40.3.23 dvmig

```
procedure(cfo_dvm), pointer cfo::zoocfo::dvmig
```

function for diel vertical migration (DVM)

Definition at line 104 of file [cfo.f90](#).

6.40.3.24 dynsvm

```
logical cfo::zoocfo::dynsvm =.FALSE.
```

dynamic traits for seasonal vertical migration

Definition at line 69 of file [cfo.f90](#).

6.40.3.25 e

```
real(dp) cfo::zoocfo::e =1._dp
```

assimilation efficiency

Definition at line 22 of file [cfo.f90](#).

6.40.3.26 egest

```
procedure(egest_pm), pointer cfo::zoocfo::egest
```

Definition at line 102 of file [cfo.f90](#).

6.40.3.27 egestion

```
character(len=512) cfo::zoocfo::egestion ='det '
```

name of compartment for egestion

Definition at line 59 of file [cfo.f90](#).

6.40.3.28 emax

```
real(dp) cfo::zoocfo::emax =0.99_dp
```

max. assimilation efficiency

Definition at line 22 of file [cfo.f90](#).

6.40.3.29 eq

```
real(dp), dimension(:), allocatable cfo::zoocfo::eq
```

particulate egestion

Definition at line 75 of file [cfo.f90](#).

6.40.3.30 excrete

```
procedure(excrete_dom), pointer cfo::zoocfo::excrete
```

Definition at line 101 of file [cfo.f90](#).

6.40.3.31 fdchl

```
real(dp), pointer cfo::zoocfo::fdchl
```

dissolved fraction of egested Chl

Definition at line 93 of file [cfo.f90](#).

6.40.3.32 fde

```
real(dp), dimension(:), allocatable cfo::zoocfo::fde
```

dissolved fraction of egestion

Definition at line 75 of file [cfo.f90](#).

6.40.3.33 fdpic

```
real(dp), pointer cfo::zoocfo::fdpic
```

dissolved fraction of egested PIC

Definition at line 93 of file [cfo.f90](#).

6.40.3.34 fhphi

```
real(dp) cfo::zoocfo::fhphi =1._dp
```

hibernation factor for phi

Definition at line 22 of file [cfo.f90](#).

6.40.3.35 fhrm

```
real(dp) cfo::zoocfo::fhrm =1._dp
```

hibernation factor for Rm

Definition at line 22 of file [cfo.f90](#).

6.40.3.36 forage

```
procedure(forage\_cfo), pointer cfo::zoocfo::forage
```

Definition at line 100 of file [cfo.f90](#).

6.40.3.37 fpx

```
real(dp), dimension(:), allocatable cfo::zoocfo::fpx
```

particulate fraction of excretion

Definition at line 75 of file [cfo.f90](#).

6.40.3.38 fsm

```
real(dp) cfo::zoocfo::fsm
```

extension factor of the migration time window

Definition at line 22 of file [cfo.f90](#).

6.40.3.39 g

```
real(dp) cfo::zoocfo::g = 0._dp
```

net growth rate

Definition at line 22 of file [cfo.f90](#).

6.40.3.40 gmax

```
real(dp) cfo::zoocfo::gmax
```

max. net growth rate

Definition at line 22 of file [cfo.f90](#).

6.40.3.41 i

```
real(dp) cfo::zoocfo::i = 0._dp
```

ingestion

Definition at line 22 of file [cfo.f90](#).

6.40.3.42 icdet

```
integer, dimension(:), allocatable cfo::zoocfo::icdet
```

Definition at line 61 of file [cfo.f90](#).

6.40.3.43 id0svm

```
integer cfo::zoocfo::id0svm = 0
```

index of summer-time level for SVM

Definition at line 63 of file [cfo.f90](#).

6.40.3.44 id0svm1

```
integer cfo::zoocfo::id0svm1
```

$\text{id0svm} + 1$

Definition at line 63 of file [cfo.f90](#).

6.40.3.45 id1dvm

```
integer cfo::zoocfo::id1dvm = 0
```

index of day-time level for DVM

Definition at line 63 of file [cfo.f90](#).

6.40.3.46 id1svm

```
integer cfo::zoocfo::id1svm = 0
```

index of winter-time level for SVM

Definition at line 63 of file [cfo.f90](#).

6.40.3.47 id1svm1

```
integer cfo::zoocfo::id1svm1
```

$\text{id1svm} - 1$

Definition at line 63 of file [cfo.f90](#).

6.40.3.48 ida

```
integer cfo::zoocfo::ida =0
```

Definition at line 63 of file [cfo.f90](#).

6.40.3.49 idd

```
integer cfo::zoocfo::idd =0
```

Definition at line 63 of file [cfo.f90](#).

6.40.3.50 idet

```
integer, dimension(:), allocatable cfo::zoocfo::idet
```

Definition at line 61 of file [cfo.f90](#).

6.40.3.51 idig

```
integer, dimension(:), allocatable cfo::zoocfo::idig
```

Definition at line 61 of file [cfo.f90](#).

6.40.3.52 idim

```
integer, dimension(:), allocatable cfo::zoocfo::idim
```

Definition at line 61 of file [cfo.f90](#).

6.40.3.53 idom

```
integer, dimension(:), pointer cfo::zoocfo::idom
```

Definition at line 62 of file [cfo.f90](#).

6.40.3.54 imaf

```
real(dp) cfo::zoocfo::imaf
```

Imax for ambush feeding.

Definition at line 22 of file [cfo.f90](#).

6.40.3.55 imax

```
real(dp) cfo::zoocfo::imax
```

max. specific ingestion rate (1/d)

Definition at line 22 of file [cfo.f90](#).

6.40.3.56 imot

```
integer, dimension(:), allocatable cfo::zoocfo::imot
```

Definition at line 61 of file [cfo.f90](#).

6.40.3.57 inot

```
integer, dimension(:), allocatable cfo::zoocfo::inot
```

Definition at line 61 of file [cfo.f90](#).

6.40.3.58 ip

```
real(dp), dimension(:), allocatable cfo::zoocfo::ip
```

individual prey loss rates (prey mortalities)

Definition at line 75 of file [cfo.f90](#).

6.40.3.59 irda

```
integer, pointer cfo::zoocfo::irda
```

Definition at line 62 of file [cfo.f90](#).

6.40.3.60 irdd

```
integer, pointer cfo::zoocfo::irdd
```

Definition at line 62 of file [cfo.f90](#).

6.40.3.61 isda

```
integer cfo::zoocfo::isda
```

Definition at line 63 of file [cfo.f90](#).

6.40.3.62 isdd

```
integer cfo::zoocfo::isdd
```

Definition at line 63 of file [cfo.f90](#).

6.40.3.63 mort

```
real(dp) cfo::zoocfo::mort =0._dp
```

fish mortality in the surface layer(s)

Definition at line 22 of file [cfo.f90](#).

6.40.3.64 ngr

```
real(dp) cfo::zoocfo::ngr
```

net growth rate

Definition at line 22 of file [cfo.f90](#).

6.40.3.65 pcc

```
type(pcc), dimension(:), allocatable cfo::zoocfo::pcc
```

prey capture coefficient associations

Definition at line 74 of file [cfo.f90](#).

6.40.3.66 phi

```
real(dp), dimension(:), allocatable cfo::zoocfo::phi
```

food preferences -> prey capture coefficients ($\text{m}^3 (\text{mol C})^{-1}$)

Definition at line 75 of file [cfo.f90](#).

6.40.3.67 phi0

```
real(dp), dimension(:), allocatable cfo::zoocfo::phi0
```

default phi for SVM

Definition at line 75 of file [cfo.f90](#).

6.40.3.68 phim

```
real(dp), dimension(:), allocatable cfo::zoocfo::phim
```

phis for motile prey

Definition at line 75 of file [cfo.f90](#).

6.40.3.69 phin

```
real(dp), dimension(:), allocatable cfo::zoocfo::phin
```

phis for non-motile prey

Definition at line 75 of file [cfo.f90](#).

6.40.3.70 pp

```
real(dp), dimension(:), allocatable cfo::zoocfo::pp
```

effective prey (food concentration*phi)

Definition at line 75 of file [cfo.f90](#).

6.40.3.71 ppm

```
real(dp), dimension(:), allocatable cfo::zoocfo::ppm
```

same as PP but for motile prey only

Definition at line 75 of file [cfo.f90](#).

6.40.3.72 ppn

```
real(dp), dimension(:), allocatable cfo::zoocfo::ppn
```

same as PP but for non-motile prey only

Definition at line 75 of file [cfo.f90](#).

6.40.3.73 pq

```
real(dp), dimension(:), allocatable cfo::zoocfo::pq
```

prey cell quotas

Definition at line 75 of file [cfo.f90](#).

6.40.3.74 pth

```
real(dp) cfo::zoocfo::pth
```

feeding threshold (mmolC m^{-3})

Definition at line 22 of file [cfo.f90](#).

6.40.3.75 qn_param

```
real(dp) cfo::zoocfo::qn_param =0.16_dp
```

Definition at line 22 of file [cfo.f90](#).

6.40.3.76 qp_param

```
real(dp) cfo::zoocfo::qp_param =0.01_dp
```

Definition at line 22 of file [cfo.f90](#).

6.40.3.77 quotas

```
real(dp), dimension(:), pointer cfo::zoocfo::quotas
```

zooplankton nutrient:C ratios

Definition at line 90 of file [cfo.f90](#).

6.40.3.78 r

```
real(dp) cfo::zoocfo::r =0._dp
```

respiration

Definition at line 22 of file [cfo.f90](#).

6.40.3.79 rff

```
real(dp), dimension(:), allocatable cfo::zoocfo::rff
```

ratio of PP/SUM(PP), corrected for assimilation efficiencies

Definition at line 75 of file [cfo.f90](#).

6.40.3.80 rffm

```
real(dp), dimension(:), pointer cfo::zoocfo::rffm
```

RFF for motile prey.

Definition at line 90 of file [cfo.f90](#).

6.40.3.81 rffn

```
real(dp), dimension(:), pointer cfo::zoocfo::rffn
```

RFF for non-motile prey.

Definition at line 90 of file [cfo.f90](#).

6.40.3.82 rm

```
real(dp) cfo::zoocfo::rm
```

specific maintenance respiration (1/d)

Definition at line 22 of file [cfo.f90](#).

6.40.3.83 rm0

```
real(dp) cfo::zoocfo::rm0 =0._dp
```

default Rm

Definition at line 22 of file [cfo.f90](#).

6.40.3.84 rq

```
real(dp), dimension(:), allocatable cfo::zoocfo::rq
```

remineralisation of dissolved inorganic nutrients

Definition at line 75 of file [cfo.f90](#).

6.40.3.85 svmig

```
procedure(cfo_svm_static), pointer cfo::zoocfo::svmig
```

function for seasonal vertical migration (SVM)

Definition at line 103 of file [cfo.f90](#).

6.40.3.86 switch

```
logical cfo::zoocfo::switch =.FALSE.
```

switching between motile/non-motile prey

Definition at line 69 of file [cfo.f90](#).

6.40.3.87 thcsvm

```
real(dp) cfo::zoocfo::thcsvm =1.E-6_dp
```

threshold for svm in mmolC m^{-3}

Definition at line 22 of file [cfo.f90](#).

6.40.3.88 toy

```
real(dp) cfo::zoocfo::toy
```

time of year (decimal days)

Definition at line 22 of file [cfo.f90](#).

6.40.3.89 v0svm

```
real(dp) cfo::zoocfo::v0svm =0._dp
```

potential seasonal vertical migration velocity (m/d)

Definition at line 22 of file [cfo.f90](#).

6.40.3.90 vdvm

```
real(dp) cfo::zoocfo::vdvm =0._dp
```

diel vertical migration velocity

Definition at line 22 of file [cfo.f90](#).

6.40.3.91 vsvm

```
real(dp), pointer cfo::zoocfo::vsvm
```

seasonal vertical migration velocity (m/d)

Definition at line 93 of file [cfo.f90](#).

6.40.3.92 vsvm0

```
real(dp) cfo::zoocfo::vsvm0 =0._dp
```

migration velocity when leaving the surface (m/s)

Definition at line 22 of file [cfo.f90](#).

6.40.3.93 vsvm1

```
real(dp) cfo::zoocfo::vsvm1 =0._dp
```

migration velocity when leaving hibernation depth (m/s)

Definition at line 22 of file [cfo.f90](#).

6.40.3.94 xq

```
real(dp), dimension(:), allocatable cfo::zoocfo::xq
```

total excretion including egestion

Definition at line 75 of file [cfo.f90](#).

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

- [/Users/mpahlow/oppla/src/cfo.f90](#)

Chapter 7

File Documentation

7.1 /Users/mpahlow/oppla/src/bac.f90 File Reference

Data Types

- type `bac::bacteria`
- type `bac::dom`

Modules

- module `bac`
functions for bacteria and DOM

Functions/Subroutines

- subroutine `bac::bac_flux` (grp, grps, env, box, times)
- subroutine `bac::bac_grow` (bac, box)
- subroutine `bac::bac_read` (grp, env, lun)
- subroutine `bac::bac_set` (grp, grps, env)
- real(dp) function `bac::bac_uptake` (bac)
saturation of DOC uptake (independent of temperature)
- real(dp) function `bac::bac_uptake_ft` (bac)
saturation of DOC uptake depends on temperature
- subroutine `bac::dom_flux` (grp, grps, env, box, times)
set DOM-related fluxes in the source matrix soma
- subroutine `bac::dom_read` (grp, env, lun)
read namelist dom and set dom parameters, constituents, number of states
- subroutine `bac::dom_set` (grp, grps, env)
flag DOM-related elements of soma for output

7.2 bac.f90

[Go to the documentation of this file.](#)

```

00001
00002 MODULE bac
00003   USE et
00004   IMPLICIT NONE
00005   PRIVATE
00006
00007   TYPE, EXTENDS(fungroup), PUBLIC :: bacteria
00008     REAL(dp) :: qn_param=0.2, qp_param=0.01,&
00009       adim=1.e0_dp,&
00010       adop=0._dp,&
00011       ftem,&
00012       ggem=0.3d0,&
00013       lambda=10._dp,&
00014       n2p,&
00015       vdin, vdip,&
00016       rc,&
00017       vm,&
00018       vmax=5d0,&
00019       zeta=0.6d0
00020     CHARACTER(LEN=100) :: dom='DOM'
00021     INTEGER :: idom=0
00022     INTEGER, POINTER :: idocnp(:)
00023     REAL(dp), ALLOCATABLE :: docnp(:), vdom(:)
00024     REAL(dp), POINTER :: doc, don, dop, vdoc, vdon, vdop, qdon, qdop
00025     PROCEDURE(bac_grow), POINTER :: grow
00026     PROCEDURE(bac_uptake), POINTER :: uptake
00027   CONTAINS
00028     PROCEDURE :: flux => bac_flux
00029     PROCEDURE :: read => bac_read
00030     PROCEDURE :: set => bac_set
00031   END TYPE bacteria
00032
00033   TYPE, EXTENDS(fungroup), PUBLIC :: dom
00034     REAL(dp) :: trlc=0d0,&
00035       trln=0d0,&
00036       ldlc=0d0,&
00037       ldln=0d0,&
00038       relac, relan
00039   CONTAINS
00040     PROCEDURE :: flux => dom_flux
00041     PROCEDURE :: read => dom_read
00042     PROCEDURE :: set => dom_set
00043   END TYPE dom
00044
00045   CONTAINS
00046
00047   SUBROUTINE bac_read (grp, env, lun)
00048     IMPLICIT NONE
00049     CLASS(bacteria), TARGET, INTENT(INOUT) :: grp
00050     TYPE(local), TARGET, INTENT(IN) :: env
00051     INTEGER, INTENT(IN) :: lun
00052     CHARACTER(LEN=512) :: DOM, species, fT="", fn, msg
00053     LOGICAL :: fTlam
00054     REAL(dp) :: Q10, ggem, sticky
00055     TYPE(quantity) :: QN, QP, Tref, Vmax, lambda, ADIM, zeta
00056     namelist /bac/ dom, species, ft, ftlam, q10, ggem, sticky,&
00057       qn, qp, tref, vmax, lambda, adim, zeta
00058     grp%nsv = 1
00059     species = "
00060     dom = grp%DOM
00061     rewind(lun)
00062     DO
00063       ftlam = .false.
00064       ggem = grp%ggem
00065       lambda = quantity(grp%lambda, 'm3 molC^-1')
00066       qn = quantity(grp%QN_param, 'molN molC^-1')
00067       qp = quantity(grp%QP_param, 'molP molC^-1')
00068       zeta = quantity(grp%zeta, 'molC molN^-1')
00069       q10 = grp%Q10
00070       sticky = grp%sticky
00071       tref = quantity(grp%Tref, 'C')
00072       vmax = quantity(grp%Vmax, 'd-1')
00073       adim = quantity(grp%ADIM, 'm3 molC-1 d-1')
00074       READ (lun, nml=bac, END=100, IOMSG=msg, ERR=900)
00075       IF (trim(species).EQ.trim(grp%name)) EXIT
00076     END DO
00077     grp%ggem = ggem
00078     grp%lambda = convert(lambda, 'm3 mmolC^-1')
00079     grp%QN_param = convert(qn, 'molN molC^-1')
00080     grp%QP_param = convert(qp, 'molP molC^-1')
00081     grp%zeta = convert(zeta, 'molC molN^-1')
00082     grp%Q10 = q10

```

```

00083     grp%sticky = sticky
00084     grp%Tref = convert(tref, 'C')
00085     grp%Vmax = convert(vmax, 's-1')
00086     grp%ADIM = convert(adim, 'm3 mmolC-1 s-1')
00087     CALL grp%ft_select (ft, 'bac_read') ! set temperature function
00088     ALLOCATE (grp%constituents(grp%nsv))
00089     grp%constituents = 'C'
00090     grp%DOM = trim(dom(1:100))
00091     grp%grow => bac_grow
00092     IF (ftlam) THEN
00093         grp%uptake => bac_uptake_ft
00094     ELSE
00095         grp%uptake => bac_uptake
00096     END IF
00097     RETURN
00098 100 CONTINUE
00099     INQUIRE (unit=lun, name=fn)
00100     WRITE (stderr, '("bac_read: missing namelist bac for species ",A," in file ",A,".")') &
00101         trim(grp%name), trim(fn)
00102     stop 1
00103 900 CONTINUE
00104     WRITE (stderr, '("bac_read:",A)') trim(msg)
00105     stop 1
00106 END SUBROUTINE bac_read
00107
00108 SUBROUTINE bac_set (grp, grps, env)
00109     IMPLICIT NONE
00110     CLASS(bacteria), TARGET, INTENT(INOUT) :: grp
00111     TYPE(state), DIMENSION(:), INTENT(IN) :: grps
00112     TYPE(local), TARGET, INTENT(INOUT) :: env
00113     INTEGER :: n, ndom
00114     DO n=1,env%nft
00115         IF (trim(grp%DOM).EQ.trim(grps(n)%var%name)) grp%idom = n
00116     END DO
00117     IF (grp%idom.EQ.0) THEN
00118         WRITE (stderr, '("bac_set: Compartment ",A," does not exist.")') trim(grp%DOM)
00119         stop 1
00120     END IF
00121     grp%idocnp => grps(grp%idom)%var%isv
00122     ndom = SIZE(grp%idocnp)
00123     ALLOCATE (grp%DOCNP(ndom), grp%VDOM(ndom))
00124     grp%QN = grp%QN_param
00125     grp%QP = grp%QP_param
00126     grp%QDON => grps(grp%idom)%var%QN
00127     grp%QDOP => grps(grp%idom)%var%QP
00128     grp%N2P = grp%QN/grp%QP
00129     grp%DOC => grp%DOCNP(1)
00130     grp%VDOC => grp%VDOM(1)
00131     IF (ndom.GT.1) THEN
00132         grp%DON => grp%DOCNP(2)
00133         grp%VDON => grp%VDOM(2)
00134     ELSE
00135         ALLOCATE (grp%DON, grp%VDON)
00136     END IF
00137     IF (ndom.GT.2) THEN
00138         grp%DOP => grp%DOCNP(3)
00139         grp%VDOP => grp%VDOM(3)
00140     ELSE
00141         ALLOCATE (grp%DOP, grp%VDOP)
00142     END IF
00143     env%smk(grp%isv(1), (/grp%idocnp, env%idic, env%io2, env%idin, env%idip/), :) = .true.
00144 END SUBROUTINE bac_set
00145
00146 SUBROUTINE bac_flux (grp, grps, env, box, times)
00147     IMPLICIT NONE
00148     CLASS(bacteria), INTENT(INOUT) :: grp
00149     TYPE(state), DIMENSION(:), INTENT(IN) :: grps
00150     TYPE(local), INTENT(INOUT) :: env
00151     TYPE(layer), INTENT(INOUT) :: box(0:)
00152     TYPE(timing), INTENT(IN) :: times
00153     grp%DOCNP = box(1)%stv(grp%idocnp)
00154     CALL grp%grow (box(1))
00155     box(1)%soma(grp%idocnp(1),grp%isv(1)) = grp%VDOC - grp%RC
00156     box(1)%soma(grp%isv(1),grp%idocnp) = -grp%VDOM
00157     box(1)%soma(grp%isv(1),env%idic) = grp%RC
00158     box(1)%soma(grp%isv(1),env%io2) = -grp%RC*env%RQ + grp%VDIN*2d0 ! O2 consumption
00159     box(1)%soma(grp%isv(1),env%idin) = -grp%VDIN
00160     box(1)%soma(grp%isv(1),env%idip) = -grp%VDIP
00161 END SUBROUTINE bac_flux
00162
00163 SUBROUTINE bac_grow (bac, box)
00164     IMPLICIT NONE
00165     CLASS(bacteria), INTENT(INOUT) :: bac
00166     TYPE(layer), INTENT(IN) :: box
00167     REAL(dp) :: CANi, APi, APo, dmdNi, dmdP, VNimx, VNipt, VPipt, VPopt
00168     bac%fTem = bac%ft(box)
00169     bac%Vm = bac%fTem*bac%Vmax

```

```

00170     bac%VDOC = bac%uptake() ! potential DOC uptake (bac_uptake or bac_uptake_ft)
00171     bac%VDON = bac%VDOC*bac%QDON
00172     cani = bac%OC*bac%ADIM*box%DIN
00173     api = bac%ADIM*box%DIP
00174     apo = bac%ADOP*bac%DOP
00175     vpipt = bac%OC*api/(api/(bac%Vm*bac%QP) + 1d0) ! potential DIP uptake
00176     vpopt = bac%OC*apo/(apo/(bac%Vm*bac%QP) + 1d0) ! potential DOP uptake via alkaline phosphatase
00177     bac%VDOP = bac%VDOC*bac%QDOP ! DOP as part of labile DOM
00178     vnimx = (vpipt + bac%VDOP + vpopt)*bac%N2P ! mini chain model: P uptake limits N assimilation
00179     vnipt = min(vnimx*cani/(cani + vnimx), vnimx - bac%VDON) ! potential DIN uptake
00180     dmdni = bac%VDOC*bac%ggem*bac%QN - bac%VDON - min(vnipt, 0d0) ! DIN demand (always >= 0)
00181     IF (dmdni.GT.(vnipt*(1d0 + bac%zeta*bac%QN))) THEN
00182         bac%VDIN = vnipt ! DIN uptake or DON remineralization (if VNipt < 0)
00183         bac%RC = bac%VDOC - (bac%VDIN + bac%VDON)/bac%QN ! respire excess DOC
00184     ELSEIF (dmdni.GT.0d0) THEN ! DIN uptake according to demand
00185         bac%VDIN = dmdni/(1d0 + bac%zeta*bac%QN) ! VNipt is always >= 0 here
00186         bac%RC = bac%VDOC*(1d0 - bac%ggem) + bac%zeta*bac%VDIN
00187     ELSE
00188         bac%VDIN = dmdni ! remineralize excess DON (dmdNi <= 0)
00189         bac%RC = bac%VDOC*(1d0 - bac%ggem)
00190     END IF
00191     dmdp = (bac%VDOC - bac%RC)*bac%QP - bac%VDOP
00192     bac%VDIP = min(dmdp, vpipt)
00193     bac%VDOP = bac%VDOP + dmdp - bac%VDIP
00194 END SUBROUTINE bac_grow
00195
00197 FUNCTION bac_uptake_ft (bac) RESULT (VC)
00198     IMPLICIT NONE
00199     CLASS(bacteria), INTENT(IN) :: bac
00200     REAL(dp) :: vc, lc
00201     lc = bac%lambda*bac%DOC
00202     vc = bac%OC*bac%Vm*lc*(1d0 - exp(-bac%fTem*lc))
00203 END FUNCTION bac_uptake_ft
00204
00206 FUNCTION bac_uptake (bac) RESULT (VC)
00207     IMPLICIT NONE
00208     CLASS(bacteria), INTENT(IN) :: bac
00209     REAL(dp) :: vc, lc
00210     lc = bac%lambda*bac%DOC
00211     vc = bac%OC*bac%Vm*lc*(1d0 - exp(-lc))
00212 END FUNCTION bac_uptake
00213
00215 SUBROUTINE dom_read (grp, env, lun)
00216     IMPLICIT NONE
00217     CLASS(dom), TARGET, INTENT(INOUT) :: grp
00218     TYPE(local), TARGET, INTENT(IN) :: env
00219     INTEGER, INTENT(IN) :: lun
00220     CHARACTER(LEN=512) :: fn, fT="", msg
00221     REAL(dp) :: Q10
00222     TYPE(quantity) :: ldlc, ldln, trlc, trln, Tref
00223     namelist /dom/ ldlc, ldln, trlc, trln, ft, q10, tref
00224     grp%nsv= 3
00225     ALLOCATE (grp%constituents(grp%nsv))
00226     grp%constituents = (/'C', 'N', 'P'/)
00227     ldlc = quantity(grp%ldlc, 'd-1 W-1 m2')
00228     ldln = quantity(grp%ldln, 'd-1 W-1 m2')
00229     trlc = quantity(grp%trlc, 'd-1')
00230     trln = quantity(grp%trln, 'd-1')
00231     tref = quantity(grp%Tref, 'C')
00232     q10 = grp%Q10
00233     rewind(lun)
00234     READ (lun, nml=dom, END=100, IOMSG=msg, ERR=900)
00235     grp%ldlc = convert(ldlc, 's-1 W-1 m2')
00236     grp%ldln = convert(ldln, 's-1 W-1 m2')
00237     grp%trlc = convert(trlc, 's-1')
00238     grp%trln = convert(trln, 's-1')
00239     grp%Tref = convert(tref, 'C')
00240     grp%Q10 = q10
00241     CALL grp%ft_select (ft, 'dom_read') ! set temperature function
00242     RETURN
00243 100 CONTINUE
00244     INQUIRE (unit=lun, name=fn)
00245     WRITE (stderr, '("Namelist dom missing in file ",A,".")') trim(fn)
00246     stop 1
00247 900 CONTINUE
00248     WRITE (stderr, '("dom_read:",A)') trim(msg)
00249     stop 1
00250 END SUBROUTINE dom_read
00251
00253 SUBROUTINE dom_set (grp, grps, env)
00254     IMPLICIT NONE
00255     CLASS(dom), TARGET, INTENT(INOUT) :: grp
00256     TYPE(state), DIMENSION(:), INTENT(IN) :: grps
00257     TYPE(local), TARGET, INTENT(INOUT) :: env
00258     env%smask(env%irdoc, grp%isv(1),:) = .true.
00259     env%smask(env%irdon, grp%isv(2),:) = .true.
00260 END SUBROUTINE dom_set

```

```

00261
00263 SUBROUTINE dom_flux (grp, grps, env, box, times)
00264   IMPLICIT NONE
00265   CLASS(dom), INTENT(INOUT) :: grp
00266   TYPE(state), DIMENSION(:), INTENT(IN) :: grps
00267   TYPE(local), INTENT(INOUT) :: env
00268   TYPE(layer), INTENT(INOUT) :: box(0:)
00269   TYPE(timing), INTENT(IN) :: times
00270   REAL(dp) :: fTem
00271   ftem = grp%FT(box(1))
00272   grp%relac = grp%trlc*ftem*box(1)%stv(env%irdoc) - grp%ldlc*box(1)%iPAR*box(1)%stv(grp%isv(1))
00273   grp%relan = grp%trln*ftem*box(1)%stv(env%irdon) - grp%ldln*box(1)%iPAR*box(1)%stv(grp%isv(2))
00274   box(1)%soma(env%irdoc,grp%isv(1)) = grp%relac ! refractory to labile DOC
00275   box(1)%soma(env%irdon,grp%isv(2)) = grp%relan ! refractory to labile DON
00276   box(1)%soma(grp%isv(1),env%irdoc) = -grp%relac
00277   box(1)%soma(grp%isv(2),env%irdon) = -grp%relan
00278 END SUBROUTINE dom_flux
00279
00280 END MODULE bac

```

7.3 /Users/mpahlow/oppla/src/brock81.f90 File Reference

Modules

- module [brock81](#)
Functions for solar radiation and the diurnal light cycle.

Functions/Subroutines

- subroutine, public [brock81::brock81_init](#) (envir, times)
Initialise the [brock81](#) module.
- real(dp) function [brock81::csrad](#) (times, hour)
clear-sky insolation at sea level
- subroutine [brock81::daylength](#) (times)
daylength in decimal days
- subroutine [brock81::declination_b81](#) (times)
solar declination after [Brock \(1981\)](#)
- subroutine [brock81::declination_f95](#) (times)
solar declination after Forsythe:1995
- subroutine [brock81::par_brock81](#) (ambient, time)
instantaneous surface irradiance
- subroutine [brock81::par_ld](#) (ambient, time)
- subroutine [brock81::par_mesoqua](#) (ambient, time)
- real(dp) function [brock81::rdrad](#) (times, hour)
ratio of actual to average daily irradiance
- subroutine [brock81::sunday_brock81](#) (times, env)
sunrise, noon, sunset and day length at times%now after [Brock \(1981\)](#)
- subroutine [brock81::sunday_const](#) (times, env)
- subroutine [brock81::sunday_ld](#) (times, env)
sunrise, noon, sunset for artificial LD cycle with constant daylength
- subroutine [brock81::sunday_mesoqua](#) (times, env)
sunrise, noon, sunset and daylength at timesnow for MesoAqua

Variables

- `real(dp)`, parameter `brock81::cfhds = 0.5_dp*cfds`
12h in s
- `real(dp)`, parameter `brock81::d15 = rad*15._dp`
15° in rad
- `real(dp)`, parameter `brock81::fd = rad*360._dp/yearDays`
conversion factor from day to rad
- `real(dp)` `brock81::sdl`
*short form for $SIN(d1)*SIN(latrad)$ (+ $SIN(ptl)$)*
- `real(dp)`, parameter `brock81::solar = 1.353E3_dp`
solar constant in W/m^2
- `real(dp)`, parameter `brock81::tilt = rad*23.45_dp`
tilt of the earth against its orbital plane in radians
- `real(dp)`, parameter `brock81::yeardays = 365.2425_dp`
length of a year in days

7.4 brock81.f90

[Go to the documentation of this file.](#)

```

00001
00005 MODULE brock81
00006   USE et
00007   USE julian
00008   IMPLICIT NONE
00009   PRIVATE
00010   PUBLIC :: brock81_init
00011   REAL(dp), PARAMETER :: d15=rad*15._dp,&
00012     solar=1.353e3_dp,&
00013     tilt=rad*23.45_dp,&
00014     yeardays=365.2425_dp,&
00015     fd=rad*360._dp/yeardays,&
00016     cfhds=0.5_dp*cfds
00017   REAL(dp) :: sdl
00018 CONTAINS
00019
00025 SUBROUTINE brock81_init (envir, times)
00026   IMPLICIT NONE
00027   TYPE(local), INTENT(INOUT) :: envir
00028   TYPE(timing), TARGET, INTENT(INOUT) :: times
00029   times%daylength => daylength
00030   IF (.NOT.ASSOCIATED(times%latrad)) THEN
00031     ALLOCATE (times%latrad)
00032     times%latrad = times%latdeg*rad ! times%latdeg from physical forcing file
00033   END IF
00034   SELECT CASE (trim(envir%daylenfun))
00035   CASE ('Brock81')
00036     times%declination => declination_b81
00037   CASE ('Forsythe95')
00038     times%declination => declination_f95
00039   CASE DEFAULT
00040     stop 'Unknown daylength function in namelist envi.'
00041   END SELECT
00042   SELECT CASE (trim(envir%dicy)) ! set diurnal light cycle
00043   CASE ('', 'none')
00044     ! constant light
00045     envir%PARfun => par_const
00046     times%sunday => sunday_const
00047     times%sun = nint(times%end, kind=int64)
00048   CASE ('Brock81', 'brock81') ! natural light cycle after Brock (1981)
00049     envir%PARfun => par_brock81
00050     times%sunday => sunday_brock81
00051   CASE ('MesoAqua', 'mesoaqua', 'MESOAQUA') ! light cycle for MesoAqua experiments
00052     ! (Lewandowska et al. 2014)
00053     envir%PARfun => par_meso_aqua
00054     times%sunday => sunday_meso_aqua
00055   CASE ('LD')
00056     ! fixed rectangular light-dark cycle
00057     envir%PARfun => par_ld
00058     times%sunday => sunday_ld
00059   CASE DEFAULT
00060     stop 'Unknown light cycle (dicy) in namelist envi.'
00061   END SELECT
00062   times%sunrise => times%sun(1)

```

```

00060     times%sunset => times%sun(3)
00061 END SUBROUTINE brock81_init
00062
00068 SUBROUTINE sunday_brock81 (times, env)
00069     IMPLICIT NONE
00070     CLASS(timing), INTENT(INOUT) :: times
00071     TYPE(local), INTENT(INOUT) :: env
00072     INTEGER(KIND=INT64) :: tsec, seconds
00073     INTEGER(KIND=4) :: dutc, year, yr, month, day
00074     INTEGER(4), SAVE :: y0=0, m0=0
00075     tsec = nint(times%now + times%tbase, kind=int64)
00076     dutc = int(tsec/cfds, kind=4)
00077     seconds = modulo(tsec, cfds) ! current time of day in seconds
00078     CALL jul_ydofdutc (dutc, yr, times%doy)
00079     CALL daylength (times)
00080     times%sun = nint(times%now + times%noon*(1._dp + (/ -times%daylen, 0._dp, times%daylen, 1._dp/)), &
00081                    kind=int64) - seconds
00082     IF (env%calcPAR) env%dayPAR = csrad(times, 12._dp)/rdrad(times, 12._dp)
00083     CALL jul_ymdofdutc (dutc, year, month, day)
00084     IF (year.NE.y0) THEN
00085         y0 = year
00086         WRITE (stdout, '(I4)', advance='NO') year
00087     END IF
00088     IF (month.NE.m0) THEN
00089         m0 = month
00090         WRITE (stdout, '(I2,":")', advance='NO') month
00091     END IF
00092     WRITE (stdout, '(I2)', advance='NO') day
00093 END SUBROUTINE sunday_brock81
00094
00096 SUBROUTINE par_brock81 (ambient, time)
00097     IMPLICIT NONE
00098     CLASS(local), INTENT(INOUT) :: ambient
00099     TYPE(timing), INTENT(INOUT) :: time
00100     ambient%sPAR = rdrad(times=time, hour=time%hour)*ambient%dayPAR
00101     entry par_const(ambient, time)
00102 END SUBROUTINE par_brock81
00103
00107 SUBROUTINE sunday_mesoauqua (times, env)
00108     IMPLICIT NONE
00109     CLASS(timing), INTENT(INOUT) :: times
00110     TYPE(local), INTENT(INOUT) :: env
00111     env%daylen = (578.4d0 + floor(times%now + times%tbase))/864d2
00112     times%sun = nint(times%now + times%noon*(1d0 + (/ -env%daylen, 0._dp, env%daylen, 1._dp/)))
00113 END SUBROUTINE sunday_mesoauqua
00114
00115 SUBROUTINE par_mesoauqua (ambient, time)
00116     IMPLICIT NONE
00117     CLASS(local), INTENT(INOUT) :: ambient
00118     TYPE(timing), INTENT(INOUT) :: time
00119     ambient%sPAR = max(ambient%dayPAR*min((1.6d0 - 3.6d0*abs(0.5d0 - time%hour/24._dp))&
00120                                         /ambient%daylen - 0.6d0, 1d0), 0d0)
00121 END SUBROUTINE par_mesoauqua
00122
00127 SUBROUTINE sunday_ld (times, env)
00128     IMPLICIT NONE
00129     CLASS(timing), INTENT(INOUT) :: times
00130     TYPE(local), INTENT(INOUT) :: env
00131     times%sun = nint(times%now - mod(times%now, 86.4e3_dp)&
00132                    + times%noon*(1._dp + (/ -env%daylen, 0._dp, env%daylen, 1._dp/)))
00133 END SUBROUTINE sunday_ld
00134
00135 SUBROUTINE par_ld (ambient, time)
00136     IMPLICIT NONE
00137     CLASS(local), INTENT(INOUT) :: ambient
00138     TYPE(timing), INTENT(INOUT) :: time
00139     IF (abs(1._dp - time%hour/12._dp).LE.ambient%daylen) THEN
00140         ambient%sPAR = ambient%dayPAR
00141     ELSE
00142         ambient%sPAR = 0._dp
00143     END IF
00144 END SUBROUTINE par_ld
00145
00146 SUBROUTINE sunday_const (times, env)
00147     IMPLICIT NONE
00148     CLASS(timing), INTENT(INOUT) :: times
00149     TYPE(local), INTENT(INOUT) :: env
00150 END SUBROUTINE sunday_const
00151
00162 FUNCTION csrad (times, hour) RESULT (it)
00163     IMPLICIT NONE ! ac0, ab0, k0 are base values for ac, ab, and k for 0 altitude
00164     REAL(dp), PARAMETER :: ac0=0.12814_dp, ab0=0.7568875_dp, k0=0.387225_dp ! (20-22)
00165     CLASS(timing), INTENT(IN) :: times
00166     REAL(dp), INTENT(IN) :: hour
00167     REAL(dp) :: sr, w2, irls, cosz, fak, ac, ab, k, tr, td, it
00168     w2 = d15*(hour - 12._dp)
00169     cosz = max(sdl + times%cdl*cos(w2), 0._dp)

```

! hour-angle
! z = zenith angle

```

00170      irls = 1._dp + 0.033_dp*cos(fd*times%day)           ! 1/r12, r1 is the radius vector
00171      sr = solar*irls*cosz                                ! rad. at top of atmosphere
00172      fak = cos(times%latrad)*(1d0 - sign(1d0, times%latrad)*times%dl) ! from Hottel (1976)?
00173      ac = ac0*(0.95_dp + 0.05_dp*fak)
00174      ab = ab0*(0.98_dp + 0.02_dp*fak)
00175      k = k0*(1.02_dp - 0.02_dp*fak)
00176      tr = ac + ab*exp(-min(k/cosz, 700d0)) ! transmission factor (19)
00177      td = 0.2710_dp - 0.2939_dp*tr          ! ratio of diffuse to extraterr. rad. (24)
00178      it = sr*(tr + td)
00179      END FUNCTION csrad
00180
00192      FUNCTION rdrad (times, hour) RESULT (dr)
00193      IMPLICIT NONE
00194      CLASS(timing), INTENT(IN) :: times
00195      REAL(dp), INTENT(IN) :: hour
00196      REAL(dp) :: w2, cosz, dr
00197      w2 = d15*(hour - 12._dp)                ! hour-angle
00198      cosz = max(sdl + times%cdl*cos(w2), 0d0) ! z = zenith angle
00199      IF (times%w1.GT.0d0) THEN
00200        dr = pi*cosz/(times%w1*(sdl + times%cdl))
00201      ELSE
00202        dr = 0d0
00203      END IF
00204      END FUNCTION rdrad
00205
00211      SUBROUTINE declination_b81 (times)
00212      IMPLICIT NONE
00213      CLASS(timing), INTENT(INOUT) :: times
00214      times%dl = tilt*sin(fd*(284._dp + times%day)) ! declination
00215      sdl = sin(times%latrad)*sin(times%dl)
00216      END SUBROUTINE declination_b81
00217
00224      SUBROUTINE declination_f95 (times)
00225      IMPLICIT NONE
00226      REAL(dp), PARAMETER :: a=0.39795_dp, b=0.2163108_dp, c=0.9671396_dp, d=pi/yeardays
00227      CLASS(timing), INTENT(INOUT) :: times
00228      times%dl = asin(a*cos(b + 2._dp*atan(c*tan(d*(times%day - 186)))) ! declination
00229      sdl = sin(times%ptl) + sin(times%latrad)*sin(times%dl)
00230      END SUBROUTINE declination_f95
00231
00238      SUBROUTINE daylength (times)
00239      IMPLICIT NONE
00240      CLASS(timing), INTENT(INOUT) :: times
00241      REAL(dp) :: cw1
00242      CALL times%declination()
00243      times%cdl = cos(times%latrad)*cos(times%dl)
00244      cw1 = -sdl/(times%cdl) ! COS(sunset hour-angle)
00245      times%w1 = acos(sign(min(abs(cw1), 1._dp), cw1)) ! sunset hour-angle
00246      times%daylen = times%w1/pi ! day length
00247      END SUBROUTINE daylength
00248      END MODULE brock81

```

7.5 /Users/mpahlow/oppla/src/cfo.f90 File Reference

Data Types

- type `cfo::pcc`

Associate prey type with prey capture coefficient. [More...](#)

- type `cfo::zoocfo`

Zooplankton

Components `depsvm`, `v0svm`, `dtsvma`, and `(doya and doyd)` or `svm` must be set to enable seasonal vertical migration.

Components `depdvm` and `vdvm` must be set to enable diel vertical migration.

Modules

- module `cfo`

Optimal current-feeding model for zooplankton.

Functions/Subroutines

- subroutine [cfo::cfo_dvm](#) (zoo, box, times)
Vertical velocity for diel vertical migration.
- subroutine [cfo::cfo_flux](#) (grp, grps, env, box, times)
flux routine for the cfo module
- subroutine [cfo::cfo_read](#) (grp, env, lun)
Read namelist cfo for matching species.
- subroutine [cfo::cfo_set](#) (grp, grps, env)
- subroutine [cfo::cfo_svm_dyn](#) (zoo, box, times)
Vertical velocity for seasonal vertical migration and days of ascent and descent.
- subroutine [cfo::cfo_svm_static](#) (zoo, box, times)
Vertical velocities and days of ascent and descent.
- subroutine [cfo::egest_dm](#) (zoo, box)
- subroutine [cfo::egest_pm](#) (zoo, box)
- subroutine [cfo::excrete_dim](#) (zoo, env, box)
- subroutine [cfo::excrete_dom](#) (zoo, env, box)
- subroutine [cfo::foract](#) (zoo, box, pc, ei, rm)
- subroutine [cfo::forage_cfo](#) (zoo, env, box)
- subroutine [cfo::forage_switch](#) (zoo, env, box)

7.5.1 Data Type Documentation

7.5.1.1 type cfo::pcc

Associate prey type with prey capture coefficient.

Definition at line 13 of file [cfo.f90](#).

Class Members

type(quantity)	phi	prey capture coefficient
character(len=100)	type	prey type, corresponds to groupname

7.6 cfo.f90

[Go to the documentation of this file.](#)

```

00001
00006 MODULE cfo
00007   USE lambert
00008   USE et
00009   IMPLICIT NONE
00010   PRIVATE
00011
00013   TYPE :: pcc
00014     CHARACTER(LEN=100) :: type
00015     TYPE(quantity) :: phi
00016   END TYPE pcc
00017
00021   TYPE, EXTENDS(fungroup), PUBLIC :: zoocfo
00022     REAL(dp) :: ngr, &           !< net growth rate
00023     imax, &                     !< max. specific ingestion rate (1/d)
00024     imaf, &                     !< Imax for ambush feeding
00025     rm, &                       !< specific maintenance respiration (1/d)
00026     rm0=0._dp, &

```

```

00027         fhrm=1._dp,&
00028         fhphi=1._dp,&
00029         beta=0.2_dp,&
00030         ca=0.1_dp,&
00031         cf0=0.1_dp, cf,&
00032         d0dvm=0._dp,&
00033         depdvm=0._dp,&
00034         dtdvm=1.5_dp,&
00035         d0svm=0._dp,&
00036         depsvm=0._dp,&
00037         dtsvma=0._dp,&
00038         dtsvmd=0._dp,&
00039         v0svm=0._dp,&
00040         vdvm=0._dp,&
00041         vsvm0=0._dp,&
00042         vsvml=0._dp,&
00043         emax=0.99_dp,&
00044         cmax=1._dp,&
00045         af=0._dp,&
00046         at,&
00047         mort=0._dp,&
00048         i=0._dp,&
00049         e=1._dp,&
00050         r=0._dp,&
00051         g=0._dp,&
00052         gmax,&
00053         pth,&
00054         thcsvm=1.e-6_dp,&
00055         fsm,&
00056         toy,&
00057         qn_param=0.16_dp,&
00058         qp_param=0.01_dp
00059 CHARACTER(LEN=512) :: egestion='det',& !< name of compartment for egestion
00060         dom='DOM'
00061 INTEGER, DIMENSION(:), ALLOCATABLE :: idet, idim, imot, inot, icdet, idig
00062 INTEGER, POINTER :: idom(:), irda, irdd
00063 INTEGER :: ida=0, idd=0, isda, isdd,&
00064         idldvm=0,&
00065         id0svm=0,&
00066         id0svml,&
00067         idlsvm=0,&
00068         idlsvml
00069 LOGICAL :: switch=.false.,&
00070         dodvm=.false.,&
00071         dodvm0,&
00072         dosvm=.false.,&
00073         dynsvm=.false.
00074 TYPE(pcc), DIMENSION(:), ALLOCATABLE :: pcc
00075 REAL(dp), DIMENSION(:), ALLOCATABLE :: fde,& !< dissolved fraction of egestion
00076         fpx,& !< particulate fraction of excretion
00077         ip,& !< individual prey loss rates (prey mortalities)
00078         phi,& !< food preferences -> prey capture coefficients (\unit{m^3.(mol.C)^{-1}})
00079         phi0,& !< default phi for SVM
00080         pq,& !< prey cell quotas
00081         rq,& !< remineralisation of dissolved inorganic nutrients
00082         pp,& !< effective prey (food concentration*phi)
00083         eq,& !< particulate egestion
00084         xq,& !< total excretion including egestion
00085         ppm,& !< same as PP but for motile prey only
00086         ppn,& !< same as PP but for non-motile prey only
00087         rff,& !< ratio of PP/SUM(PP), corrected for assimilation efficiencies
00088         phim,& !< phis for motile prey
00089         phin
00090 REAL(dp), DIMENSION(:), POINTER :: rffm,& !< RFF for motile prey
00091         rffn,& !< RFF for non-motile prey
00092         quotas
00093 REAL(dp), POINTER :: cxda,& !< product of biomass and doya
00094         fdpic,& !< dissolved fraction of egested PIC
00095         fdchl,& !< dissolved fraction of egested Chl
00096         vsvm,& !< seasonal vertical migration velocity (m/d)
00097         doya,& !< day of year of ascent
00098         doyd,& !< day of year of descent
00099         cxdd
00100 PROCEDURE(forage_cfo), POINTER :: forage
00101 PROCEDURE(excrete_dom), POINTER :: excrete
00102 PROCEDURE(egest_pm), POINTER :: egest
00103 PROCEDURE(cfo_svm_static), POINTER :: svmig
00104 PROCEDURE(cfo_dvm), POINTER :: dvmig
00105 CONTAINS
00106 PROCEDURE :: flux => cfo_flux
00107 PROCEDURE :: read => cfo_read
00108 PROCEDURE :: set => cfo_set
00109 END TYPE zoocfo
00110
00111 CONTAINS
00112
00114 SUBROUTINE cfo_read (grp, env, lun)

```

```

00115      IMPLICIT NONE
00116      CLASS(zocfo), TARGET, INTENT(INOUT) :: grp
00117      TYPE(local), TARGET, INTENT(IN) :: env
00118      INTEGER, INTENT(IN) :: lun
00119      CHARACTER(LEN=512) :: species, DOM, fn, egest, forage, fT, msg, svm
00120      TYPE(pcc), DIMENSION(:), ALLOCATABLE :: phi
00121      TYPE(quantity) :: Imax, Rm, Cmax, QN, QP, Tref, mort, d0svm, depsvm, dtsvm,&
00122          dtsvma, dtsvmd, vsvm, doya, doyd, thcsvm, d0dvm, depdvm, dtdvm, vdvm
00123      REAL(dp) :: Emax, beta, ca, cf, px, q10, fde, fdPIC, feChl, fhRM, fhphi
00124      LOGICAL :: motile, digest_Ch1
00125      namelist /cfo/ species, beta, ca, cf, emax, phi, imax, rm, cmax, qn, qp,&
00126          px, motile, forage, egest, ft, dom, q10, tref, fde, fdpic, fechl, doya, doyd,&
00127          d0svm, depsvm, dtsvm, dtsvma, dtsvmd, vsvm, svm, thcsvm,&
00128          d0dvm, depdvm, dtdvm, vdvm, fhrm, fhphi, digest_ch1, mort
00129      ALLOCATE (phi(env%nft+1), grp%phi(env%nft), grp%phi0(env%nft), grp%pcc(env%nft),&
00130          grp%IP(env%nft), grp%PP(env%nft), grp%constituents(1),&
00131          grp%fde(env%ncon), grp%fpX(env%ncon))
00132      grp%constituents = 'C'
00133      grp%fdPIC => grp%fde(4)
00134      grp%fdCh1 => grp%fde(5)
00135      grp%IP = 0d0
00136      dom = grp%DOM
00137      species = "
00138      rewind(lun)
00139      DO
00140          ft = "
00141          egest = grp%egestion
00142          forage = 'cfo'
00143          motile = .true.
00144          beta = grp%beta
00145          ca = grp%ca
00146          cf = grp%cf0
00147          emax = grp%Emax
00148          cmax = quantity(grp%Cmax, 'm3 molC-1')
00149          phi = pcc("", quantity(0d0, 'm3 molC-1'))
00150          imax = quantity(grp%Imax, 'd-1')
00151          rm = quantity(grp%Rm0, 'd-1')
00152          qn = quantity(grp%QN_param, 'molN molC-1')
00153          qp = quantity(grp%QP_param, 'molP molC-1')
00154          tref = quantity(grp%Tref, '°C')
00155          q10 = grp%Q10
00156          fde = 0._dp
00157          fdpic = 0._dp
00158          fechl = 1._dp
00159          fhrm = grp%fhRm
00160          fhphi = grp%fhphi
00161          digest_ch1 = .true.
00162          mort = quantity(grp%mort, 'd-1')
00163          doyd = quantity(-1._dp, 'd')
00164          doya = quantity(-1._dp, 'd')
00165          d0svm = quantity(grp%d0svm, 'm')
00166          depsvm = quantity(grp%depsvm, 'm')
00167          dtsvm = quantity(0._dp, 'd')
00168          dtsvma = quantity(grp%dtsvma, 'd')
00169          dtsvmd = quantity(grp%dtsvmd, 'd')
00170          vsvm = quantity(grp%v0svm, 'm d-1')
00171          thcsvm = quantity(grp%thcsvm, 'mmolC m-3')
00172          svm = "
00173          d0dvm = quantity(grp%d0dvm, 'm')
00174          depdvm = quantity(grp%depdvm, 'm')
00175          dtdvm = quantity(grp%dtdvm, 'h')
00176          vdvm = quantity(grp%vdvm, 'm d-1')
00177          READ (lun, nml=cfo, END=100, IOMSG=msg, ERR=900)
00178          IF (len_trim(species).GT.100) &
00179              WRITE (*,('Warning: truncating species to ",A100)') species(1:100)
00180          IF (trim(species(1:100)).EQ.trim(grp%name)) EXIT
00181      END DO
00182      grp%beta = beta
00183      grp%ca = ca
00184      grp%cf0 = cf
00185      grp%Emax = emax
00186      grp%Cmax = convert(cmax, 'm3 molC-1')
00187      grp%Imax = convert(imax, 's-1')
00188      grp%Rm0 = convert(rm, 's-1')
00189      grp%Rm = grp%Rm0
00190      grp%QN_param = convert(qn, 'molN molC-1')
00191      grp%QP_param = convert(qp, 'molP molC-1')
00192      grp%Q10 = q10
00193      grp%fde = fde
00194      grp%fdPIC = fdpic
00195      grp%fdCh1 = 1._dp - fechl
00196      grp%fhrm = fhrm
00197      grp%fhphi = fhphi
00198      grp%Tref = convert(tref, '°C')
00199      IF (phi(env%nft+1)%phi%value.GT.0._dp) THEN
00200          INQUIRE (unit=lun, name=fn)
00201          WRITE (stderr,('cfo_read: too many entries for phi in namelist cfo in file ",A,")') trim(fn)

```

```

00202      stop 1
00203      END IF
00204      phi(1:env%nft)%phi%value = phi(1:env%nft)%phi%value*grp%Cmax
00205      grp%pcc = phi(1:env%nft)
00206      grp%fpv = 1._dp
00207      grp%DOM = dom
00208      grp%motile = motile
00209      grp%d0svm = convert(d0svm, 'm')
00210      grp%depsvm = convert(depsvm, 'm')
00211      IF ((dtsvm%value.GT.0._dp).AND.(dtsvma%value.LE.0._dp)) dtsvma = dtsvm
00212      IF ((dtsvm%value.GT.0._dp).AND.(dtsvmd%value.LE.0._dp)) dtsvmd = dtsvm
00213      grp%dtsvma = convert(dtsvma, 'd')
00214      grp%dtsvmd = convert(dtsvmd, 'd')
00215      grp%mort = convert(mort, 's-1')
00216      grp%v0svm = convert(vsvm, 'm s-1')
00217      grp%thcsvm = convert(thcsvm, 'mmolC m-3')
00218      grp%dosvm = (grp%depsvm.GT.0._dp).AND.(grp%v0svm.GT.0._dp).AND.(grp%dtsvma.GT.0._dp) &
00219      .AND.(((doya%value.GT.0._dp).AND.(doyd%value.GT.0._dp)).OR.(trim(svm).EQ.'dynamic'))
00220      grp%d0dvm = convert(d0dvm, 'm')
00221      grp%depdvm = convert(depdm, 'm')
00222      grp%dtddvm = convert(dtddvm, 's')
00223      grp%vddvm = convert(vddvm, 'm s-1')
00224      grp%dodvm = (grp%depdvm.GT.0._dp).AND.(grp%vddvm.GT.0._dp)
00225      grp%dodvm0 = grp%dodvm ! default setting, needed for combining DVM and SVM
00226      grp%egestion = trim(egest(1:100))
00227      ALLOCATE(grp%idig(3+count((/digest_ch1/)))
00228      grp%idig(1:3) = (/1, 2, 3/)
00229      IF (digest_ch1) grp%idig(4) = 5 ! digest Ch1 in parallel with C
00230      SELECT CASE (trim(forage))
00231      CASE ('switch')
00232          grp%switch = .true.
00233          grp%forage => forage_switch
00234      !      grp%At = grp%Imax/grp%beta*(-1D0 - lwml(-EXP(-1D0 - grp%beta))) ! (E21)
00235          grp%At = grp%Imax & ! total activity (15)
00236          *(-1d0 - lwml(-(1d0 - grp%cf0/grp%Emax/(1d0 - grp%ca))/exp(1d0 + grp%beta)))
00237          grp%Imaf = grp%Imax
00238      !      grp%Imaf = grp%At*grp%beta/(-1D0 - lwml(-EXP(-1D0 - grp%beta))) ! (E21)
00239      CASE ('cfo')
00240          grp%switch = .false.
00241          grp%forage => forage_cfo
00242          grp%cf = grp%cf0
00243          grp%At = grp%Imax & ! total activity (15)
00244          *(-1d0 - lwml(-(1d0 - grp%cf/grp%Emax/(1d0 - grp%ca))/exp(1d0 + grp%beta)))
00245          grp%Pth = -log((1d0 - grp%cf/grp%Emax/(1d0 - grp%ca)))*(1._dp + 1.e-15_dp) ! /grp%phi
00246      CASE DEFAULT
00247          stop 'cfo_read: Invalid choice for forage in namelist cfo'
00248      END SELECT
00249      grp%nsv = 1
00250      IF (grp%dosvm) THEN
00251          SELECT CASE (trim(svm))
00252          CASE ('static', '')
00253              ALLOCATE (grp%doya, grp%doyd)
00254              grp%doya = convert(doya, 'd')
00255              grp%doyd = convert(doyd, 'd')
00256              grp%svmig => cfo_svm_static
00257              grp%vsvm => grp%v0svm
00258              grp%dtsvma = 0.5_dp*grp%dtsvma
00259              grp%dtsvmd = 0.5_dp*grp%dtsvmd
00260          CASE ('dynamic')
00261              grp%nsv = 3
00262              grp%ida = 2
00263              grp%idd = 3
00264              grp%svmig => cfo_svm_dyn
00265              grp%dynsvm = .true.
00266              ALLOCATE (grp%vsvm)
00267              grp%vsvm = grp%v0svm
00268          CASE DEFAULT
00269              stop 'cfo_read: Invalid choice for svm in namelist cfo'
00270          END SELECT
00271      ELSE
00272          grp%svmig => cfo_nosvm
00273      END IF
00274      IF (grp%dodvm) grp%dvdmig => cfo_dvm
00275      CALL grp%ft_select (ft, 'cfo_read')
00276      RETURN
00277 100 CONTINUE
00278      INQUIRE (unit=lun, name=fn)
00279      WRITE(stderr,fmt='("cfo/cfo_read: missing namelist cfo for species ",A," in file ",A,".")')&
00280      trim(grp%name), trim(fn)
00281      stop 1
00282 900 CONTINUE
00283      WRITE (stderr,'("cfo_read:",A)') trim(msg)
00284      stop 1
00285  END SUBROUTINE cfo_read
00286
00287  SUBROUTINE cfo_set (grp, grps, env)
00288      IMPLICIT NONE

```

```

00289     CLASS(zocfo), TARGET, INTENT(INOUT) :: grp
00290     TYPE(state), DIMENSION(:), INTENT(IN) :: grps
00291     TYPE(local), TARGET, INTENT(INOUT) :: env
00292     INTEGER :: n, itype(1), idep(1), nm, nn
00293     grp%quotas => env%quotas(:,grp%ifg)
00294     grp%QN = grp%QN_param
00295     grp%QP = grp%QP_param
00296     ALLOCATE (grp%PQ(env%ncon), grp%RQ(env%ncon), grp%EQ(env%ncon), grp%XQ(env%ncon),
    grp%idim(env%ncon))
00297     grp%idim = (/env%idic, env%idin, env%idip/)
00298     NULLIFY (grp%idom, grp%egest)
00299     grp%phi = 0d0
00300     IF (grp%switch) THEN ! switch between motile and non-motile prey
00301         nm = count((/grps(n)%var%motile, n=1,env%nft/)) ! No. of motile types
00302         nn = env%nft - nm ! No. of non-motile types
00303         ALLOCATE (grp%RFF(env%nft), grp%imot(nm), grp%PPm(nm), grp%phim(nm), &
00304             grp%inot(nn), grp%PPn(nn), grp%phin(nn))
00305         grp%RFFm => grp%RFF(1:nm)
00306         grp%RFFn => grp%RFF(nm+1:nm+nn)
00307         nm = 0
00308         nn = 0
00309     END IF
00310     IF (grp%dosvm) THEN
00311         idep = minloc(env%depth_edges(2,:), mask=env%depth_edges(2,:).GE.grp%depsvm)
00312         grp%idlsvm = min(idep(1), env%nbox) ! index of winter-time level
00313         IF (grp%dynsvm) THEN ! dynamic traits (days of ascent and descent) for svm
00314             grp%idlsvm1 = grp%idlsvm - 1 ! index of winter-time level - 1
00315             grp%doya => grp%ratios(1)
00316             grp%doyd => grp%ratios(2)
00317             grp%irda => grp%ir(1) ! doya in box n is box(n)%ratios(grp%irda)
00318             grp%irdd => grp%ir(2)
00319             grp%isda = grp%isv(grp%ida)
00320             grp%isdd = grp%isv(grp%idd)
00321             ! traits are stored as tracer-trait products
00322             grp%names(2) = trim(grp%name)//'_CxDa' ! product of biomass (C) and doya
00323             grp%names(3) = trim(grp%name)//'_CxDd' ! product of biomass (C) and doyd
00324             grp%units(2:3) = 'd '//trim(grp%units(1))
00325             idep = maxloc(env%depth_edges(1,:), mask=env%depth_edges(1,:).LE.grp%d0svm)
00326             grp%id0svm = idep(1) ! index of summer-time level
00327             grp%id0svm1 = grp%id0svm + 1 ! index of summer-time level + 1
00328             grp%vsvsm0 = 10._dp*env%depth_edges(2,grp%id0svm)/grp%dtsvmd/cfds
00329             grp%vsvsm1 = 10._dp*env%heights(1,grp%idlsvm)/grp%dtsvma/cfds
00330             grp%fsvm = grp%depsvm/grp%vsvsm/grp%dtsvma/cfds + 1._dp
00331         END IF
00332     END IF
00333     IF (grp%dodvm) THEN
00334         idep = minloc(env%depth_edges(2,:), mask=env%depth_edges(2,:).GE.grp%depdvm)
00335         grp%idlsvm = min(idep(1), env%nbox) ! index of day-time level
00336         idep = maxloc(env%depth_edges(1,:), mask=env%depth_edges(1,:).LE.grp%d0dvm)
00337         grp%d0dvm = max(grp%d0dvm, env%depth(1)) ! no upward motion in the surface layer
00338     END IF
00339     env%smask(grp%isv(1), (/grp%idim, env%io2/), :) = .true.
00340     DO n=1,env%nft
00341         IF (any(grps(n)%var%name.EQ.grp%pcc%type)) THEN ! associate phi with prey type
00342             itype = pack([nn, nn = 1, env%nft], mask=grps(n)%var%name.EQ.grp%pcc%type)
00343             grp%phi(n) = convert(grp%pcc(itype(1))%phi, 'm3 mmolC-1')
00344         END IF
00345         IF (trim(grp%egestion).EQ.trim(grps(n)%var%name)) THEN ! egest particulate matter to detritus
00346             ALLOCATE(grp%idet(grps(n)%var%ncon), grp%icdet(grps(n)%var%ncon))
00347             grp%icdet = grps(n)%var%icon
00348             grp%idet = grps(n)%var%isv
00349             grp%egest => egest_pm
00350         END IF
00351         IF (trim(grp%DOM).EQ.trim(grps(n)%var%name)) grp%idom => grps(n)%var%isv
00352         IF (grp%switch) THEN
00353             IF (grps(n)%var%motile) THEN
00354                 nm = nm + 1
00355                 grp%imot(nm) = n
00356                 grp%phim(nm) = grp%phi(n)
00357             ELSE
00358                 nn = nn + 1
00359                 grp%inot(nn) = n
00360                 grp%phin(nn) = grp%phi(n)
00361             END IF
00362         END IF
00363         env%smask(grp%isv(1), grps(n)%var%isv, :) = .true.
00364     END DO
00365     IF (.NOT.ASSOCIATED(grp%egest)) grp%egest => egest_dm
00366     grp%phi0 = grp%phi
00367     IF (ASSOCIATED(grp%idom)) THEN
00368         grp%excrete => excrete_dom
00369         env%smask(grp%isv(1), grp%idom, :) = .true.
00370     ELSE
00371         grp%excrete => excrete_dim
00372         env%smask(grp%isv(1), grp%idim, :) = .true.
00373     END IF
00374 END SUBROUTINE cfo_set

```

```

00375
00377 SUBROUTINE cfo_flux (grp, grps, env, box, times)
00378   IMPLICIT NONE
00379   CLASS(zoo_cfo), INTENT(INOUT) :: grp
00380   TYPE(state), DIMENSION(:), INTENT(IN) :: grps
00381   TYPE(local), INTENT(INOUT) :: env
00382   TYPE(layer), INTENT(INOUT) :: box(0:)
00383   TYPE(timing), INTENT(IN) :: times
00384   INTEGER :: n
00385   IF (box(1)%nb.EQ.grp%idlsvm) THEN
00386     grp%Rm = grp%fhRm*grp%Rm0 ! reduced maintenance loss at hibernation depth
00387     grp%phi = grp%fhphi*grp%phi0 ! no feeding activity at hibernation depth
00388   ELSE
00389     grp%Rm = grp%Rm0
00390     grp%phi = grp%phi0
00391   END IF
00392   CALL grp%forage (env, box(1))
00393   CALL grp%svmig (box=box, times=times) ! seasonal vertical migration (SVM)
00394   IF (grp%dodvm) CALL grp%dvmig (box=box(1), times=times) ! diel vertical migration (DVM)
00395   ! store all fluxes in row isv(1) of the flux matrix
00396   box(1)%soma(grp%isv(1),grp%idim) = grp%RQ(1:3) ! release of inorganic nutrients (DIC, DIN, DIP)
00397   DO n = 1,env%nft
00398     box(1)%soma(grp%isv(1),grps(n)%var%isv) = -grp%IP(n)*box(1)%stv(grps(n)%var%isv)
00399   END DO
00400   box(1)%soma(grp%isv(1),grp%isv(1)) = box(1)%soma(grp%isv(1),grp%isv(1)) + grp%ngr
00401   CALL grp%excrete (env, box(1))
00402   CALL grp%egest (box(1))
00403   ! add the dissolved fraction of released PIC to DIC
00404   box(1)%soma(grp%isv(1),grp%idim(1)) = box(1)%soma(grp%isv(1),grp%idim(1)) + grp%fdPIC*grp%EQ(4)
00405   box(1)%soma(grp%isv(1),env%io2) = -grp%R*(env%RQ + grp%QN*2d0) ! O2 consumption
00406 END SUBROUTINE cfo_flux
00407
00419 SUBROUTINE cfo_dvm (zoo, box, times)
00420   IMPLICIT NONE
00421   CLASS(zoo_cfo), INTENT(INOUT) :: zoo
00422   TYPE(layer), INTENT(IN) :: box
00423   TYPE(timing), INTENT(IN) :: times
00424   REAL(dp) :: xdown, xup
00425   IF (box%nb.LT.zoo%idldvm) THEN
00426     xdown = max(zoo%dtldvm - abs(times%current - times%sunrise), 0._dp)**8
00427     xup = max(zoo%dtldvm - abs(times%current - times%sunset), 0._dp)**8
00428   ELSE
00429     xdown = 0._dp
00430     xup = 0._dp
00431   END IF
00432   zoo%vv = zoo%vldvm*(xdown/(xdown + cfd0) - &
00433     xup/(xup + cfd0)*min(2._dp*max(1._dp - zoo%id0dvm/box%depth, 0._dp), 1._dp))
00434 END SUBROUTINE cfo_dvm
00435
00452 SUBROUTINE cfo_svm_dyn (zoo, box, times)
00453   IMPLICIT NONE
00454   CLASS(zoo_cfo), INTENT(INOUT) :: zoo
00455   TYPE(layer), INTENT(IN) :: box(0:)
00456   TYPE(timing), INTENT(IN) :: times
00457   REAL(dp), SAVE :: fda, fdd=0._dp
00458   ! Zooplankton growth should have no effect on doya and doyd, so the
00459   ! corresponding trait-tracer products must change at the same relative rate
00460   ! (ngr) as zooplankton biomass.
00461   box(1)%soma(zoo%isda,zoo%isda) = zoo%ngr*zoo%doya ! (2.14)
00462   box(1)%soma(zoo%isdd,zoo%isdd) = zoo%ngr*zoo%doyd
00463   ! When the animals leave their target svm layers (boxes), id0svm in summer
00464   ! and idlsvm in winter, they take the current time with them as their doya
00465   ! (for the ascent) or doyd (for the descent). This is implemented here by
00466   ! modifying the vertical velocity of the tracer-trait products within the
00467   ! surface and hibernation ranges during migration, so that effectively the
00468   ! SVM traits transport the current time of year.
00469   IF (box(1)%nb.LT.zoo%idlsvm) THEN ! surface and migration ranges
00470     fda = (times%toy - box(2)%ratios(zoo%irda))/zoo%dtsvma + 0.5_dp
00471     IF ((fda.GT.0._dp).AND.(fda.LE.zoo%fsvm)) THEN ! ascent
00472       IF (box(1)%nb.LE.zoo%id0svm) THEN ! surface range
00473         zoo%vv = -0.5_dp*zoo%vsvm0
00474       ELSEIF (box(1)%nb.LT.zoo%idlsvm1) THEN ! migration range
00475         zoo%vv = -zoo%vsvm
00476       ELSEIF ((box(1)%nb.EQ.zoo%idlsvm1).AND.(fda.LE.1._dp)) THEN ! above hibernation layer
00477         zoo%vv = -fda*zoo%vsvm1 ! velocity out of hibernation layer
00478         zoo%vv(zoo%ida) = times%toy/box(2)%ratios(zoo%irda)*zoo%vv(1)
00479       ELSE
00480         zoo%vv = 0._dp
00481       END IF
00482     ELSE
00483       ! the fdd used here is from the previous call of this subroutine, one
00484       ! layer above
00485       IF ((box(1)%nb.EQ.zoo%id0svm1).AND.(fdd.GT.0._dp).AND.(fdd.LE.1.0_dp)) THEN
00486         box(0)%soma(zoo%isdd,zoo%isdd) = box(0)%soma(zoo%isdd,zoo%isdd) &
00487           + box(0)%bbc(zoo%isdd)*(times%toy/box(0)%ratios(zoo%irda) - 1._dp)&
00488           *fdd*zoo%vsvm0/box(0)%height
00489       END IF

```

```

00490      fdd = (times%toy - zoo%doyd)/zoo%dtsvmd + 0.5_dp
00491      IF ((fdd.GT.0._dp).AND.(fdd.LE.zoo%fsvm)) THEN ! descent
00492      IF (box(1)%nb.GT.zoo%id0svm) THEN ! migration range
00493      zoo%vv = zoo%vsvm
00494      ELSEIF (fdd.LE.1._dp) THEN ! surface range
00495      zoo%vv = fdd*zoo%vsvm0
00496      IF (box(1)%nb.EQ.zoo%id0svm) zoo%vv(zoo%idd) = times%toy/zoo%doyd*zoo%vv(1)
00497      ELSE
00498      zoo%vv = 0._dp
00499      END IF
00500      ELSE
00501      zoo%vv = 0._dp
00502      END IF
00503      END IF
00504      ELSEIF (box(1)%nb.EQ.zoo%idlsvm) THEN ! hibernation layer
00505      fda = (times%toy - zoo%doya)/zoo%dtsvma + 0.5_dp
00506      IF ((fda.GT.0._dp).AND.(fda.LE.1._dp)) THEN ! ascent
00507      box(1)%soma(zoo%isda,zoo%isda) = box(1)%soma(zoo%isda,zoo%isda) &
00508      + box(0)%bbc(zoo%isda)*(times%toy/zoo%doya - 1._dp)&
00509      *fda*zoo%vsvml/box(1)%height
00510      END IF
00511      zoo%vv = 0._dp
00512      ELSE
00513      zoo%vv = 0._dp
00514      END IF
00515      zoo%dodvm = zoo%dodvm0.AND.& ! no DVM during hibernation
00516      (times%toy.GT.(zoo%doya + zoo%dtsvma)).AND.(times%toy.LT.(zoo%doyd - zoo%dtsvmd))
00517      END SUBROUTINE cfo_svm_dyn
00518
00531      SUBROUTINE cfo_svm_static (zoo, box, times)
00532      IMPLICIT NONE
00533      CLASS(zoo), INTENT(INOUT) :: zoo
00534      TYPE(layer), INTENT(IN) :: box(0:)
00535      TYPE(timing), INTENT(IN) :: times
00536      REAL(dp) :: fdoy
00537      ! Svm motions start at doya/doyd - dtsvm and end at doya/doyd + dtsvm.
00538      ! Svm ascent slows down before reaching d0svm and is about 0 above.
00539      ! The function is based on the symmetric part of (6) in Moisan et al. (2002).
00540      fdoy = ((times%toy - zoo%doyd)/zoo%dtsvmd)**8 + (box(1)%depth/zoo%depsvm)**16
00541      zoo%vv = 0._dp
00542      IF (fdoy.LT.200._dp) THEN
00543      zoo%vv = zoo%vsvm*exp(-fdoy) ! descent (positive)
00544      ELSE
00545      fdoy = ((times%toy - zoo%doya)/zoo%dtsvma)**8 &
00546      + ((box(1)%depth - zoo%depsvm - zoo%d0svm)/(zoo%depsvm - zoo%d0svm/2._dp))**4
00547      IF (fdoy.LT.200._dp) zoo%vv = -zoo%vsvm*exp(-fdoy) ! ascent (negative)
00548      END IF
00549      IF (box(1)%nb.EQ.zoo%idlsvm) THEN
00550      zoo%Rm = zoo%fhRm*zoo%Rm0 ! reduced maintenance loss at hibernation depth
00551      zoo%phi = zoo%fhphi*zoo%phi0 ! no feeding activity at hibernation depth
00552      ELSE
00553      zoo%Rm = zoo%Rm0
00554      zoo%phi = zoo%phi0
00555      END IF
00556      zoo%dodvm = zoo%dodvm0.AND.& ! no DVM during hibernation
00557      (times%toy.GT.(zoo%doya + zoo%dtsvma)).AND.(times%toy.LT.(zoo%doyd - zoo%dtsvmd))
00558      entry cfo_nosvm(zoo, box, times)
00559      END SUBROUTINE cfo_svm_static
00560
00561      SUBROUTINE excrete_dom (zoo, env, box)
00562      IMPLICIT NONE
00563      CLASS(zoo), INTENT(INOUT) :: zoo
00564      TYPE(local), INTENT(INOUT) :: env
00565      TYPE(layer), INTENT(INOUT) :: box
00566      REAL(dp) :: add_DOC
00567      box%soma(zoo%isv(1),zoo%idom) = zoo%EQ(1:3)*zoo%fde(1:3)
00568      ! mobilize refractory DOC by adsorption of freshly-excreted DON, in proportion to the
00569      ! refractory DOC:DON ratio:
00570      add_doc = zoo%EQ(2)*zoo%fde(2)*box%stv(env%irdoc)/max(box%stv(env%irdon), 1.e-30_dp)
00571      box%soma(env%irdoc,zoo%idom(1)) = box%soma(env%irdoc,zoo%idom(1)) + add_doc
00572      box%soma(zoo%idom(1),env%irdoc) = box%soma(zoo%idom(1),env%irdoc) - add_doc
00573      END SUBROUTINE excrete_dom
00574
00575      SUBROUTINE excrete_dim (zoo, env, box)
00576      IMPLICIT NONE
00577      CLASS(zoo), INTENT(INOUT) :: zoo
00578      TYPE(local), INTENT(INOUT) :: env
00579      TYPE(layer), INTENT(INOUT) :: box
00580      ! PIC release is accounted for in cfo_flux and there is no inorganic Chl
00581      box%soma(zoo%isv(1),zoo%idim) = box%soma(zoo%isv(1),zoo%idim) + zoo%EQ(1:3)*zoo%fde(1:3)
00582      END SUBROUTINE excrete_dim
00583
00584      SUBROUTINE egest_dm (zoo, box)
00585      IMPLICIT NONE
00586      CLASS(zoo), INTENT(IN) :: zoo
00587      TYPE(layer), INTENT(INOUT) :: box
00588      ! add egested matter to associated inorganic nutrients

```

```

00589     box%soma(zoo%isv(1),zoo%idim) = box%soma(zoo%isv(1),zoo%idim) + zoo%EQ(1:3)*(1d0 - zoo%fde(1:3))
00590 END SUBROUTINE egest_dm
00591
00592 SUBROUTINE egest_pm (zoo, box)
00593     IMPLICIT NONE
00594     CLASS(zoo%fo), INTENT(IN) :: zoo
00595     TYPE(layer), INTENT(INOUT) :: box
00596     ! add egested particles to associated detritus
00597     box%soma(zoo%isv(1),zoo%idet) = box%soma(zoo%isv(1),zoo%idet) &
00598         + zoo%EQ(zoo%icdet)*(1d0 - zoo%fde(zoo%icdet))
00599 END SUBROUTINE egest_pm
00600
00601 SUBROUTINE forage_cfo (zoo, env, box)
00602     IMPLICIT NONE
00603     CLASS(zoo%fo), INTENT(INOUT) :: zoo
00604     TYPE(local), INTENT(IN) :: env
00605     TYPE(layer), INTENT(IN) :: box
00606     REAL(dp) :: Rm, fQ, EI, SPP, XC, mort
00607     zoo%PP = zoo%phi*env%OC ! effective prey (food concentration*phi)
00608     spp = max(sum(zoo%PP), 1.e-30_dp)
00609     zoo%PQ = matmul(env%quotas, zoo%PP/spp) ! prey cell quotas
00610     ! fQ accounts for different N:C ratios in predator and prey, after
00611     ! Kjørboe (1989). C Ingestion is unaffected by food C:N ratio according
00612     ! to Kjørboe (1989), so Af should not be affected either. Thus, food
00613     ! C:N should influence only R and ngr. The same concept is applied here
00614     ! also to C:P.
00615     fq = 1._dp
00616     IF (spp.GT.1.e-30_dp) fq = minval(zoo%PQ(1:3)/zoo%quotas(1:3))
00617     ! If it is assumed that the additional respiration can be used to increase
00618     ! the prey capture coefficient, this increases the effective strength of the
00619     ! feeding current. The choice of the functional relation (fphi) is arbitrary
00620     ! as no information is available other than that N growth efficiency and
00621     ! maximum ingestion should not be affected.
00622     ! fphi = EXP(1D0 - fQ)
00623     CALL foract (zoo, box, pc=spp, ei=ei, rm=rm)
00624     mort = box%fish*zoo%mort*zoo%OC ! fish mortality in the surface layer(s)
00625     zoo%ngr = (ei*(1d0 - zoo%ca) - zoo%cf*zoo%Af - rm)*fq - mort ! net growth (9)
00626     zoo%R = ei - zoo%ngr - mort ! respiration (2)
00627     xc = zoo%I - ei + mort ! particulate egestion (faecal pellets) + fish mortality
00628     zoo%fpX(zoo%idig) = xc/max(xc + zoo%R, 1d-50) ! ratio of egestion : metabolic losses
00629     zoo%XQ = zoo%I*zoo%PQ - zoo%ngr*zoo%quotas
00630     zoo%RQ = zoo%XQ*(1._dp - zoo%fpX) ! remineralisation
00631     zoo%EQ = zoo%XQ+zoo%fpX ! particulate egestion
00632     zoo%IP = zoo%I*zoo%phi/spp ! prey loss rates in 1/s
00633 END SUBROUTINE forage_cfo
00634
00635 SUBROUTINE forage_switch (zoo, env, box)
00636     IMPLICIT NONE
00637     CLASS(zoo%fo), INTENT(INOUT) :: zoo
00638     TYPE(local), INTENT(IN) :: env
00639     TYPE(layer), INTENT(IN) :: box
00640     REAL(dp) :: I0m, Rm, fQ, Em, EI, EIm, EIn, SPPm, SPPn, Im, In, XC
00641     zoo%PP = zoo%phi*env%OC ! effective prey (food concentration*phi)
00642     zoo%PPm = zoo%PP(zoo%imot) ! effective motile prey
00643     zoo%PPn = zoo%PP(zoo%inot) ! effective non-motile prey
00644     sppm = sum(zoo%PPm)
00645     sppn = sum(zoo%PPn)
00646     i0m = 1d0 - exp(-sppm) ! ingestion saturation for motile prey
00647     em = zoo%Emax*(1d0 - exp(-zoo%beta*(1d0/max(i0m, 1d-4) - 1d0)))
00648     zoo%cf = zoo%cf0 + em*i0m*(1d0 - zoo%ca)
00649     zoo%Pth = -log((1d0 - zoo%cf/zoo%Emax/(1d0 - zoo%ca)))*(1._dp + 1.e-15_dp) ! /zoo%phi
00650     CALL foract (zoo, box, sppn, ein, rm) ! current-feeding activity
00651     in = zoo%I
00652     im = (zoo%Imaf - zoo%Af)*i0m ! motile-prey ingestion (E15)
00653     zoo%I = in + im
00654     eim = em*im
00655     ei = ein + eim
00656     zoo%E = ei/zoo%I
00657     ! fQ accounts for different N:C ratios in predator and ingested prey, after
00658     ! Kjørboe (1989). C Ingestion is unaffected by food C:N ratio according
00659     ! to Kjørboe (1989), so Af should not be affected either. Thus, food
00660     ! C:N should influence only R and ngr. The same concept is applied here
00661     ! also to C:P.
00662     zoo%RFFm = eim*zoo%PPm/sppm/max(ei, 1d-30)
00663     zoo%RFFn = ein*zoo%PPn/sppn/max(ei, 1d-30)
00664     zoo%PQ = matmul(env%quotas(:, (/zoo%imot, zoo%inot/)), zoo%RFF)
00665     fq = minval(zoo%PQ(1:3)/zoo%quotas(1:3))
00666     zoo%IP(zoo%imot) = im*zoo%phi/sppm ! motile prey loss rates in 1/s
00667     zoo%IP(zoo%inot) = in*zoo%phi/sppn ! non-motile prey loss rates in 1/s
00668     zoo%ngr = (ei*(1d0 - zoo%ca) - zoo%cf0*zoo%Af - rm)*fq ! net growth (2.2)
00669     zoo%R = ei - zoo%ngr ! respiration (2.8)
00670     xc = zoo%I - ei ! particulate egestion (faecal pellets)
00671     zoo%fpX(zoo%idig) = xc/max(xc + zoo%R, 1d-50) ! ratio of egestion : metabolic losses
00672     zoo%XQ = zoo%I*zoo%PQ - zoo%ngr*zoo%quotas
00673     zoo%RQ = zoo%XQ*(1._dp - zoo%fpX) ! remineralisation
00674     zoo%EQ = zoo%XQ+zoo%fpX ! particulate egestion
00675 END SUBROUTINE forage_switch

```



```

00676
00677 SUBROUTINE foract (zoo, box, PC, EI, Rm)
00678   IMPLICIT NONE
00679   CLASS(zoo_cfo), INTENT(INOUT) :: zoo
00680   TYPE(layer), INTENT(IN) :: box
00681   REAL(dp), INTENT(IN) :: PC
00682   REAL(dp), INTENT(OUT) :: EI, Rm
00683   REAL(dp) :: At, fTC, I0, rAtf
00684   ftc = zoo%OC*zoo%fT(box)
00685   rm = zoo%Rm*ftc
00686   IF (pc.LE.zoo%Pth) THEN      ! food below feeding threshold
00687     zoo%Af = 0d0
00688     zoo%E = zoo%Emax
00689     zoo%I = 0d0
00690   ELSE
00691     i0 = 1d0 - exp(-pc)        ! ingestion saturation for non-motile prey (2.3)
00692     at = ftc*zoo%At            ! temperature dependence
00693     ! rAtf is the ratio At/Af; this is needed to prevent division by 0 when
00694     ! zoo%OC = 0 in the calculation of zoo%E; \beta was removed from At above
00695     ratf = (-1d0 - lwml*(-1d0 - zoo%cf/(zoo%Emax*(1 - zoo%ca)*i0)) & ! (2.7)
00696            /exp(1d0 + zoo%beta))
00697     ! Af: foraging activity
00698     zoo%Af = at/ratf
00699     ! E: assimilation efficiency; note that \beta was removed from At above
00700     zoo%E = zoo%Emax*(1d0 - exp(zoo%beta - ratf)) ! (2.2)
00701     zoo%I = i0*zoo%Af          ! non-motile ingestion (E14)
00702   END IF
00703   ei = zoo%E*zoo%I
00704 END SUBROUTINE foract
00705
00706 END MODULE cfo

```

7.7 /Users/mpahlow/oppla/src/clops.f90 File Reference

Modules

- module [clops](#)
process command-line options

Functions/Subroutines

- subroutine [clops::getopt](#) (opts, args, flags, usage, help)
sort arguments according to an option string

7.8 clops.f90

[Go to the documentation of this file.](#)

```

00001
00002 MODULE clops
00003   IMPLICIT NONE
00004   CONTAINS
00020 SUBROUTINE getopt (opts, args, flags, usage, help)
00021   IMPLICIT NONE
00022   CHARACTER(LEN=*) , INTENT(IN) :: opts
00023   CHARACTER(LEN=*) , INTENT(IN) :: usage
00024   CHARACTER(LEN=*) , INTENT(IN), OPTIONAL :: help
00025   CHARACTER(LEN=*) , DIMENSION(:), INTENT(INOUT) :: args
00026   LOGICAL, DIMENSION(:), INTENT(OUT) :: flags
00027   INTEGER :: i, l, nopts, no, nargc, nparg, npargs, mxlen
00028   CHARACTER (LEN=LEN(args)) :: argv, opt
00029   LOGICAL :: cov=.true.
00030   mxlen = len(args)
00031   flags = .false.
00032   nparg = 0
00033   no = 0
00034   nargc = command_argument_count()
00035   nopts = max(scan(opts, ':') - 1, 0) ! number of arguments with values
00036   npargs = SIZE(args) - max(nopts,0) ! max. number of positional arguments
00037   DO WHILE (no.LT.nargc)

```

```

00038      no = no + 1
00039      CALL get_command_argument (number=no, value=argv)
00040      IF (cov.AND.(argv(1:1).EQ.'-')) THEN
00041        i = scan(opts, argv(2:2))
00042        IF (i.GT.nopts) THEN ! flag argument (no value)
00043          flags(i-nopts-1) = .true.
00044        ELSEIF (i.GT.0) THEN
00045          l = len_trim(argv) - 2
00046          IF (l.EQ.0) THEN
00047            no = no + 1 ! next argument is the value
00048            CALL get_command_argument (number=no, value=opt)
00049            l = min(len_trim(opt), mxlen)
00050          ELSE ! value attached to option letter
00051            opt = argv(3:l+2)
00052          END IF
00053          args(i) = opt(1:l)
00054        ELSE
00055          IF (scan(argv(2:2),'hH?').GT.0) THEN
00056            IF (PRESENT(help)) THEN
00057              WRITE (*,help) usage//' | -?'; stop
00058            ELSE
00059              WRITE (*,' ("Usage: ",A," | -?")') usage
00060            END IF
00061          END IF
00062          cov = argv(2:2).NE.'-' ! -- stops option processing; remaining
00063          IF (cov) THEN ! arguments will be appended to args
00064            WRITE (*,' ("Unrecognized option ",A2,"./"Usage: ",A," | -?")')&
00065              argv(1:2), usage
00066            stop 1
00067          END IF
00068        END IF
00069      ELSEIF (nparg.LT.npargs) THEN
00070        l = min(len_trim(argv), mxlen)
00071        nparg = nparg + 1
00072        args(nparg+nopts) = argv(1:l)
00073      ELSE
00074        WRITE (*,' ("Usage: ",A)') usage//' | -?'; stop 1
00075      END IF
00076    END DO
00077    IF (nparg.GT.npargs) THEN
00078      WRITE (*,' ("Usage: ",A)') usage//' | -?'; stop 1
00079    END IF
00080  END SUBROUTINE getopt
00081 END MODULE clops

```

7.9 /Users/mpahlow/oppla/src/cmo.f90 File Reference

Data Types

- type [cmo::phycmo](#)
Ordinary phytoplankton and diazotrophs.

Modules

- module [cmo](#)
Chain-model of optimal phytoplankton growth and diazotrophy.

Functions/Subroutines

- subroutine [cmo::chl_dyn](#) (phy, box)
- subroutine [cmo::cmo_flux](#) (grp, grps, env, box, times)
- subroutine [cmo::cmo_pic](#) (phy, box)
- subroutine [cmo::cmo_read](#) (grp, env, lun)
Read parameters for optimality-based chain model.
- subroutine [cmo::cmo_set](#) (grp, grps, env)
- subroutine [cmo::facn2f](#) (phy, box, v0)

- real(dp) function `cmo::ftnf_eppley` (phy, box)
- real(dp) function `cmo::ftnf_houlton` (phy, box)
- real(dp) function `cmo::ftnf_oppla` (phy, box)
- subroutine `cmo::grow` (phy, ambient, box)
Rates of growth, nutrient uptake, and chlorophyll synthesis or steady-state Chl:C ratio.
- subroutine `cmo::tchdyn` (phy, ambient, box)
Dynamic photo-acclimation via Chl synthesis.
- subroutine `cmo::tchia` (phy, ambient, box)
Instantaneous photo-acclimation to daytime-average light intensity.
- subroutine `cmo::uptake` (phy, box, v0)

7.10 cmo.f90

[Go to the documentation of this file.](#)

```

00001
00004 MODULE cmo
00005   USE logarithmic_integral, ONLY : inloin, dei
00006   USE lambert
00007   USE et
00008   IMPLICIT NONE
00009   PRIVATE
00013   TYPE, EXTENDS(fungroup), PUBLIC :: phycmo
00014     REAL(dp) :: a0, &          !< nutrient affinity in \unit{m^3.mol^{-1}.d^{-1}}
00015     alpha, &                  !< light affinity in \unit{m^2.d^{-1}.mol.W^{-1}.gChl^{-1}}
00016     alphas, &                 !< temperature-dependent alpha
00017     qsn=0.02, &               !< structure-bound N quota in \unit{mol.mol^{-1}}
00018     rdl=1.0, &
00019     rc=0.1_dp, &
00020     rct, &
00021     v0=5._dp, &
00022     f0n=0._dp, &
00023     zc, &
00024     zn, &
00025     znu=0.6_dp, &
00026     znf=2._dp, &
00027     a, &
00028     mu0, &
00029     vph0, &
00030     svph, &
00031     fv, &
00032     fn, &
00033     fc, &
00034     ftemp, &
00035     fpic=0._dp, &
00036     ff=0._dp, &
00037     si, &
00038     tch, &
00039     rctht, &
00040     chl0=-1.0_dp, &
00041     vn, vp, &
00042     n2f=0._dp, &
00043     vpilc, &
00044     vsink=0._dp, &
00045     fcdtdq=0._dp
00046     REAL(dp), POINTER :: daylen, & !< daylength as a fraction of 24h
00047     dlfa, & !< day length factor for A
00048     par, & !< irradiance in \unit{mol.m^{-2}.d^{-1}}
00049     q0n, q0p, & !< subsistence N, P quotas
00050     qpilc, & !< PIC:POC ratio
00051     chl, & !< Chl concentration in \unit{gChl.m^{-3}}
00052     ftalpha, & !< temperature factor for alpha
00053     ftrc, & !< temperature factor for RC
00054     theta, & !< Chl:C ratio in \unit{gChl.mol^{-1}}
00055     vdoc, & !< DOC loss (\unit{VDOC \le 0}) under unbalanced growth
00056     ngr, & !< rate of N, P, C acquisition
00057     vdic
00058     REAL(dp), ALLOCATABLE :: vc(:)
00059     CHARACTER(LEN=100) :: dom='DOM'
00060     INTEGER, ALLOCATABLE :: ic(:)
00061     INTEGER :: ichl, idoc, ipic
00062     LOGICAL :: dynamicchl=.true., pic=.false.
00063     PROCEDURE(ftnf_eppley), POINTER :: ftnf => ftnf_eppley
00064     PROCEDURE(grow), POINTER :: growth
00065     PROCEDURE(uptake), POINTER :: nuts
00066     PROCEDURE(chl_dyn), POINTER :: chldyn

```

```

00067     PROCEDURE(tchdyn), POINTER :: pachl
00068     PROCEDURE(cmo_pic), POINTER :: calc
00069     CONTAINS
00070     PROCEDURE :: flux => cmo_flux
00071     PROCEDURE :: read => cmo_read
00072     PROCEDURE :: set => cmo_set
00073 END TYPE phycmo
00074
00075 CONTAINS
00076
00077 SUBROUTINE cmo_read (grp, env, lun)
00078     IMPLICIT NONE
00079     CLASS(phycmo), TARGET, INTENT(INOUT) :: grp
00080     TYPE(local), TARGET, INTENT(IN) :: env
00081     INTEGER, INTENT(IN) :: lun
00082     CHARACTER(LEN=512) :: DOM, species, msg
00083     REAL(dp) :: rdl, fF, fPIC, Q10
00084     TYPE(quantity) :: A0, alpha, Q0N, Q0P, RC, V0, F0N, zC, zN, zF,&
00085         Tref, Topt, sticky, chl0, vsink
00086     CHARACTER(LEN=15) :: pa, fT, FTNF
00087     INTEGER :: ncon
00088     LOGICAL :: fTRC, fTalpha, PIC
00089     namelist /cmo/ a0, alpha, q0n, q0p, rdl, rc, v0, f0n, zc, zn, zf, ff, sticky,&
00090         tref, topt, pa, fpic, ft, ftnf, q10, dom, ftrc, ftalpha, pic, species, vsink
00091     CALL inloin ! initialise dei function from logarithmic integral module
00092     species = "
00093     dom = grp%DOM
00094     rewind(lun)
00095     DO
00096         a0 = quantity(grp%A0, 'm3 molC-1 d-1')
00097         alpha = quantity(grp%alpha, 'm2 molC W-1 d-1 gChl-1')
00098         q0n = quantity(2._dp*grp%QsN, 'molN molC-1')
00099         q0p = quantity(0.001_dp, 'molP molC-1')
00100         rdl = grp%rdl
00101         ff = grp%ff
00102         sticky = quantity(grp%sticky, "")
00103         rc = quantity(grp%RC, 'd-1')
00104         v0 = quantity(grp%V0, 'd-1')
00105         f0n = quantity(grp%F0N, 'molN molC-1 d-1')
00106         zc = quantity(grp%zC, 'molC gChl-1')
00107         zn = quantity(grp%zNu, 'molC molN-1')
00108         zf = quantity(grp%zNf, 'molC molN-1')
00109         tref = quantity(grp%Tref, '°C')
00110         topt = quantity(grp%Topt, '°C')
00111         chl0 = quantity(grp%chl0, 'mg m-3')
00112         vsink = quantity(grp%vsink, 'm d-1')
00113         fpic = grp%fPIC
00114         q10 = grp%Q10
00115         pa = 'dynamic'
00116         ft = "
00117         ftnf = "
00118         ftrc = .true.
00119         ftalpha = .false.
00120         pic = .false.
00121         READ (lun, nml=cmo, END=100, IOMSG=msg, ERR=900)
00122         IF (trim(species(1:100)).EQ.trim(grp%name)) EXIT
00123     END DO
00124     grp%A0 = convert(a0, 'm3 mmolC-1 s-1')
00125     grp%alpha = convert(alpha, 'm2 molC W-1 s-1 gChl-1')
00126     grp%QsN = 0.5_dp*convert(q0n, 'molN molC-1')
00127     grp%rdl = rdl
00128     grp%ff = ff
00129     grp%sticky = sticky%value
00130     grp%RC = convert(rc, 's-1')
00131     grp%V0 = convert(v0, 's-1')
00132     grp%F0N = convert(f0n, 'molN molC-1 s-1')
00133     grp%zC = convert(zc, 'molC gChl-1')
00134     grp%zNu = convert(zn, 'molC molN-1')
00135     grp%zNf = convert(zf, 'molC molN-1')
00136     grp%Tref = convert(tref, '°C')
00137     grp%Topt = convert(topt, '°C')
00138     grp%chl0 = convert(chl0, 'mg m-3')
00139     grp%vsink = convert(vsink, 'm s-1')
00140     grp%Q10 = q10
00141     grp%fPIC = fpic
00142     grp%daylen => env%daylen
00143     grp%DOM = trim(dom(1:100))
00144     SELECT CASE (trim(pa))
00145     CASE ('dynamic')
00146         grp%pachl => tchdyn
00147         grp%chldyn => chl_dyn
00148     CASE ('slow')
00149         grp%pachl => tchdyn
00150         grp%chldyn => chl_dyn_slow
00151     CASE DEFAULT
00152         grp%pachl => tchia
00153         grp%chldyn => chl_dummy
00154

```

```

00155     grp%dynamicChl = .false.
00156 END SELECT
00157 grp%PIC = pic
00158 ncon = 3 + count((/grp%PIC, grp%dynamicChl/))
00159 grp%nsv = ncon
00160 grp%growth => grow
00161 IF (grp%ff.GT.0) THEN
00162   grp%nuts => facn2f
00163   IF (grp%FON.LE.0._dp) &
00164     WRITE (*, '("cmo_read: N2 fixation is on but FON = ", ES10.3, ".")') grp%FON
00165 ELSE
00166   grp%nuts => uptake
00167   grp%zN = grp%zNu
00168 END IF
00169 grp%ng0 = 2
00170 ALLOCATE (grp%Q0(grp%ng0), grp%iq0(grp%ng0), grp%constituents(grp%nsv))
00171 grp%iq0 = (/2, 3/)
00172 grp%constituents(1:3) = (/ 'C', 'N', 'P' /)
00173 IF (grp%PIC) grp%constituents(4) = 'PIC'
00174 IF (grp%dynamicChl) grp%constituents(ncon) = 'Chl'
00175 grp%Q0N => grp%Q0(1)
00176 grp%Q0P => grp%Q0(2)
00177 grp%Q0N = convert(q0n, 'molN molC-1')
00178 grp%Q0P = convert(q0p, 'molP molC-1')
00179 CALL grp%ft_select (ft, 'cmo_read')      ! set temperature function
00180 SELECT CASE (ftnf)
00181 CASE ('Eppley')
00182   grp%ftNF => ftnf_eppley
00183 CASE ('Houlton')
00184   grp%ftNF => ftnf_houlton
00185 CASE ('oppla')
00186   grp%ftNF => ftnf_oppla
00187 END SELECT
00188 IF (ftalpha) THEN
00189   grp%fTalpha => grp%fTemp
00190 ELSE
00191   ALLOCATE (grp%fTalpha)
00192   grp%fTalpha = 1._dp
00193 END IF
00194 IF (ftrc) THEN
00195   grp%fTRC => grp%fTemp
00196 ELSE
00197   ALLOCATE (grp%fTRC)
00198   grp%fTRC = 1._dp
00199 END IF
00200 RETURN
00201 100 CONTINUE
00202 WRITE (stderr, fmt= '("cmo/cmo_read: missing namelist cmo for species ", A, ".")') trim(grp%name)
00203 stop 1
00204 900 CONTINUE
00205 WRITE (stderr, '("cmo_read:", A)') trim(msg)
00206 stop 1
00207 END SUBROUTINE cmo_read
00208
00209 SUBROUTINE cmo_set (grp, grps, env)
00210 IMPLICIT NONE
00211 CLASS(phycmo), TARGET, INTENT(INOUT) :: grp
00212 TYPE(state), DIMENSION(:), INTENT(IN) :: grps
00213 TYPE(local), TARGET, INTENT(INOUT) :: env
00214 INTEGER :: n
00215 IF (grp%dynamicChl) THEN
00216   grp%theta => grp%ratios(grp%nsv-1)
00217 ELSE
00218   ALLOCATE (grp%theta)
00219 END IF
00220 SELECT CASE (trim(env%dic)) ! diurnal light cycle
00221 CASE ('', 'none') ! no light cycle (constant light)
00222   grp%dlfA => env%daylen ! calculate daily average growth
00223 CASE DEFAULT ! in this case the light cycle is resolved
00224   ALLOCATE (grp%dlfA)
00225   grp%dlfA = 1._dp ! calculate instantaneous growth rate
00226 END SELECT
00227 grp%idoc = 0
00228 DO n=1, env%nft
00229   IF (trim(grp%DOM).EQ.trim(grps(n)%var%name)) grp%idoc = grps(n)%var%isv(1)
00230 END DO
00231 IF (grp%idoc.EQ.0) THEN
00232   ALLOCATE (grp%VDIC, grp%VDOC, grp%ic(1), grp%VC(1))
00233   grp%ngr => grp%VC(1) ! growth = net C assimilation
00234 ELSE
00235   ALLOCATE (grp%ngr, grp%ic(2), grp%VC(2))
00236   grp%ic(2) = grp%idoc ! DOC release
00237   grp%VDIC => grp%VC(1) ! CO2 fixation
00238   grp%VDOC => grp%VC(2) ! DOC uptake (always .LE. 0)
00239 END IF
00240 grp%ic(1) = env%idic
00241 IF (grp%PIC) THEN

```

```

00242     grp%QPIC => grp%ratios(3)
00243     grp%calc => cmo_pic
00244     grp%ipic = grp%isv(4)
00245     grp%ichl = grp%isv(5)
00246     env%smask(grp%ic(1),grp%ipic,:) = .true.
00247 ELSE
00248     grp%calc => cmo_pic_dummy
00249     grp%ichl = grp%isv(4)
00250 END IF
00251 env%smask(grp%ic,grp%isv(1),:) = .true.
00252 env%smask(grp%isv(1),env%io2,:) = .true.
00253 env%smask(grp%isv(2),env%idin,:) = .true.
00254 env%smask(grp%isv(3),env%idip,:) = .true.
00255 IF (.NOT.ASSOCIATED(grp%stick)) ALLOCATE (grp%stick)
00256 END SUBROUTINE cmo_set
00257
00258 SUBROUTINE cmo_flux (grp, grps, env, box, times)
00259 IMPLICIT NONE
00260 CLASS(phycmo), INTENT(INOUT) :: grp
00261 TYPE(state), DIMENSION(:), INTENT(IN) :: grps
00262 TYPE(local), INTENT(INOUT) :: env
00263 TYPE(layer), INTENT(INOUT) :: box(0:)
00264 TYPE(timing), INTENT(IN) :: times
00265 CALL grp%growth (ambient=env, box=box(1))
00266 CALL grp%calc (box=box(1))
00267 grp%stick = grp%sticky*max(2._dp - grp%QN/grp%Q0N, 0._dp) ! points to env%sticky
00268 grp%vv = grp%vsink*(2._dp - min(grp%QN/grp%Q0N, 3._dp))
00269 box(1)%soma(grp%isv(1),grp%ic) = -grp%VC ! net DIC, DOC uptake
00270 box(1)%soma(grp%ic,grp%isv(1)) = grp%VC
00271 box(1)%soma(grp%isv(2),env%idin) = -grp%VN ! DIN uptake
00272 box(1)%soma(grp%isv(3),env%idip) = -grp%VP ! DIP uptake
00273 box(1)%soma(grp%isv(1),env%io2) = grp%VDIC*env%RQ + grp%VN*2._dp ! O2 production
00274 box(1)%soma(env%idin,grp%isv(2)) = grp%VN - grp%ngr*grp%Q0N ! subtract subsistence N
00275 box(1)%soma(env%idip,grp%isv(3)) = grp%VP - grp%ngr*grp%Q0P ! subtract subsistence P
00276 END SUBROUTINE cmo_flux
00277
00312 SUBROUTINE grow (phy, ambient, box)
00313 IMPLICIT NONE
00314 CLASS(phycmo), INTENT(INOUT) :: phy
00315 TYPE(local), INTENT(IN) :: ambient
00316 TYPE(layer), INTENT(IN) :: box
00317 REAL(dp) :: V0
00318 ! numbers in parentheses refer to equations in ../doc/equations.pdf
00319 phy%fTemp = phy%fT(box)
00320 v0 = phy%V0*phy%fTemp
00321 phy%alphaT = phy%alpha*phy%fTalpha
00322 phy%RCT = phy%RC*phy%fTRC
00323 phy%fV = max(phy%QsN/phy%QN - phy%zN*(phy%QN - phy%Q0N), 0._dp) ! (1.23)
00324 phy%fC = 1._dp - phy%fV - phy%QsN/phy%QN
00325 phy%mu0 = v0/(0.5_dp + ambient*daylen)*phy%rdl
00326 CALL phy%pachl (ambient, box) ! photo-acclimation (tchdyn or tchia)
00327 phy%A = (1._dp - ambient%ice)*phy%dlfA*phy%mu0*phy%SI*(1._dp - phy%zC*phy%tch) & ! (1.4)
00328 - phy%RCT*phy%zC*phy%tch
00329 CALL phy%nuts (box, v0) ! uptake of facn2f
00330 phy%VP = phy%OC*phy%fV*(1._dp - phy%fN)*phy%VPh0 ! (1.28)
00331 phy%VDIC = phy%OC*phy%fC*phy%A - phy%zN*(phy%VN + phy%N2F) ! (1.3)
00332 ! release (DO)C to avoid outgrowing Q0P
00333 phy%VDOC = -max(phy%VDIC*phy%Q0P/phy%QP - phy%VP/phy%Q0P, 0._dp)&
00334 *max(2._dp - phy%QP/phy%Q0P, 0._dp)
00335 phy%ngr = phy%VDIC + phy%VDOC
00336 CALL phy%chldyn (box) ! chl_dyn, chl_dyn_slow, or chl_dummy
00337 END SUBROUTINE grow
00338
00339 SUBROUTINE uptake (phy, box, V0)
00340 IMPLICIT NONE
00341 CLASS(phycmo), INTENT(INOUT) :: phy
00342 TYPE(layer), INTENT(IN) :: box
00343 REAL(dp), INTENT(IN) :: V0
00344 REAL(dp) :: an, ap, VNmax, Vnh0
00345 vnmax = v0*max(1._dp - phy%Q0P/phy%QP, 0._dp) ! (1.30)
00346 an = phy%A0*box%DIN
00347 ap = phy%A0*box%DIP
00348 vnh0 = an/(sqrt(max(an, 1.e-30_dp)) + sqrt(vnmax))*2 ! Vnh0/VNmax (1.29)
00349 phy%VPh0 = ap/(sqrt(ap/v0) + 1._dp)**2 ! (1.29)
00350 phy%VPh = sqrt(phy%VPh0)
00351 phy%fN = phy%VPh/(max(phy%VPh, 1.e-30_dp) + sqrt(v0*vnmax**1.5*phy%Q0P/phy%QN)) ! (1.39)
00352 phy%VN = phy%OC*phy%fV*phy%fN*vnh0*vnmax ! (1.28)
00353 END SUBROUTINE uptake
00354
00355 SUBROUTINE facn2f (phy, box, V0)
00356 IMPLICIT NONE
00357 CLASS(phycmo), INTENT(INOUT) :: phy
00358 TYPE(layer), INTENT(IN) :: box
00359 REAL(dp), INTENT(IN) :: V0
00360 REAL(dp) :: FNmax, fnF, fvF
00361 CALL uptake (phy, box, v0)
00362 fnmax = phy%F0N*phy%fTNF(box=box)

```

```

00363     fnf = phy%svPh/(sqrt(fnmax*phy%Q0P/phy%QN) + phy%svPh) ! (1.56)
00364     fvf = max(phy%QsN/phy%QN - phy%zNf*(phy%QN - phy%Q0N), 0._dp)
00365     phy%N2F = phy%OC*fvf*fnf*(1 - phy%Q0P/phy%QP)*fnmax
00366     IF (phy%VN.LT.phy%N2F) THEN ! fix N2
00367         phy%zN = phy%zNf
00368         phy%fN = fnf
00369         phy%fV = fvf
00370         phy%VN = 0._dp
00371     ELSE ! use DIN
00372         phy%zN = phy%zNu
00373         phy%N2F = 0._dp
00374     END IF
00375     box%soma(phy%isv(2),phy%isv(2)) = phy%N2F
00376 END SUBROUTINE facn2f
00377
00378 SUBROUTINE chl_dyn (phy, box)
00379     IMPLICIT NONE
00380     CLASS(phycmo), INTENT(INOUT) :: phy
00381     TYPE(layer), INTENT(IN) :: box
00382     ! ngr .GE. 0 prevents positive feedback between high theta and Chl synthesis
00383     ! fcdtdq = fC*dot(QN)/theta*dtheta/dQN (1.15)
00384     phy%fcdtdq = (phy%VN + phy%N2F - max(phy%ngr, 0._dp)*phy%QN)*(phy%Q0N/phy%QN**2 + phy%zN)
00385     entry chl_dyn_slow(phy, box)
00386     ! Chl synthesis (1.13)
00387     box%soma(phy%ichl,phy%ichl) = phy%theta*(phy%ngr + phy%rctht*phy%OC + phy%fcdtdq)
00388     entry chl_dummy(phy, box)
00389 END SUBROUTINE chl_dyn
00390
00392 SUBROUTINE tchdyn (phy, ambient, box)
00393     IMPLICIT NONE
00394     CLASS(phycmo), INTENT(INOUT) :: phy
00395     TYPE(local), INTENT(IN) :: ambient
00396     TYPE(layer), INTENT(IN) :: box
00397     REAL(dp) :: atm, atmb, atmd, SId
00398     phy%tch = phy%theta/max(phy%fC, 1.e-30_dp) ! (1.4)
00399     atm = phy%alphaT*phy%tch/max(phy%mu0, 1.e-30_dp)
00400     atmb = atm*box%bPAR
00401     IF (atmb.GT.0._dp) THEN
00402         phy%SI = 1._dp - (dei(-atm*box%tPAR) - dei(-atmb))/box%lch ! (1.62)
00403     ELSE
00404         phy%SI = 0._dp
00405     END IF
00406     ! use approx. daytime average SI (SId) for photo-acclimation, assuming a
00407     ! triangular light cycle
00408     atmd = atm*2._dp*box%dPAR ! atmd is 0 if Chl is
00409     sid = (exp(-atmd) - 1._dp + atmd)/max(atmd, 1.e-30_dp)
00410     phy%rctht = (ambient%daylen*(phy%alphaT*box%dPAR*(1._dp - sid)&
00411         *(1._dp/phy%zC - phy%tch) - sid*phy%mu0)&
00412         - phy%RCT)*phy%fC ! (fC/zC)*d(rgr)/d(theta)
00413 END SUBROUTINE tchdyn
00414
00416 SUBROUTINE tchia (phy, ambient, box)
00417     IMPLICIT NONE
00418     CLASS(phycmo), INTENT(INOUT) :: phy
00419     TYPE(local), INTENT(IN) :: ambient
00420     TYPE(layer), INTENT(IN) :: box
00421     REAL(dp) :: aim, atm, I0
00422     aim = phy%alphaT*box%dPAR/phy%mu0
00423     i0 = phy%RCT*phy%zC/(phy%alphaT*ambient%daylen) ! threshold light intensity (1.12)
00424     IF (box%dPAR.GT.i0) THEN ! steady-state Chl:C ratio (1.11)
00425         phy%tch = 1._dp/phy%zC + (1._dp - lambertw((1._dp + phy%RCT/(ambient%daylen*phy%mu0))&
00426             *exp(1._dp + aim/phy%zC), 0, 0))/aim
00427         atm = phy%tch*phy%alphaT/phy%mu0
00428         phy%SI = 1._dp - (dei(-atm*box%tPAR) - dei(-atm*box%bPAR))/box%lch
00429     ELSE ! below light threshold: no Chl, no growth
00430         phy%tch = 0._dp
00431         phy%SI = 0._dp
00432     END IF
00433     phy%theta = phy%tch*phy%fC
00434 END SUBROUTINE tchia
00435
00436 SUBROUTINE cmo_pic (phy, box)
00437     IMPLICIT NONE
00438     CLASS(phycmo), INTENT(INOUT) :: phy
00439     TYPE(layer), INTENT(INOUT) :: box
00440     phy%VPIC = phy%fPIC*phy%VDIC
00441     box%soma(phy%ipic(1),phy%ipic) = phy%VPIC ! calcification
00442     box%soma(phy%ipic,phy%ic(1)) = -phy%VPIC
00443     entry cmo_pic_dummy(phy, box)
00444 END SUBROUTINE cmo_pic
00445
00446 FUNCTION ftnf_oppla (phy, box) RESULT (ft)
00447     IMPLICIT NONE
00448     CLASS(phycmo), INTENT(IN) :: phy
00449     TYPE(layer), INTENT(IN) :: box
00450     REAL(dp) :: ft
00451     ft = ft_oppla(phy, box)

```

```

00452 END FUNCTION ftnf_oppla
00453
00454 FUNCTION ftnf_eppley (phy, box) RESULT (ft)
00455   IMPLICIT NONE
00456   CLASS(phycmo), INTENT(IN) :: phy
00457   TYPE(layer), INTENT(IN) :: box
00458   REAL(dp) :: ft
00459   ft = ft_eppley(phy, box)
00460 END FUNCTION ftnf_eppley
00461
00462 FUNCTION ftnf_houlton (phy, box) RESULT (ft)
00463   IMPLICIT NONE
00464   CLASS(phycmo), INTENT(IN) :: phy
00465   TYPE(layer), INTENT(IN) :: box
00466   REAL(dp) :: ft
00467   ft = ft_houlton(phy, box)
00468 END FUNCTION ftnf_houlton
00469 END MODULE cmo

```

7.11 /Users/mpahlow/oppla/src/det.f90 File Reference

Data Types

- type [det::detritus](#)

Modules

- module [det](#)
detritus functions

Functions/Subroutines

- subroutine [det::det_flux](#) (grp, grps, env, box, times)
- subroutine [det::det_fluxes](#) (grp, box)
- subroutine [det::det_loss_pic](#) (grp, box)
- subroutine [det::det_read](#) (grp, env, lun)
read detritus-related parameters, set detritus parameters, constituents, number of states
- subroutine [det::det_set](#) (grp, grps, env)
flag detritus fluxes in soma for output, set-up remineralisation scheme

7.12 det.f90

[Go to the documentation of this file.](#)

```

00001
00002 MODULE det
00003   USE et
00004   IMPLICIT NONE
00005   PRIVATE
00006
00007   TYPE, EXTENDS(fungroup), PUBLIC :: detritus
00008     REAL(dp) :: sink=0_dp, &
00009       omax=15._dp, &
00010       remic=0._dp, remin=0._dp, remip=0._dp, &
00011       remipic=-1._dp, &
00012       formpic=0._dp, &
00013       remichl=-1._dp, &
00014       kag=-1._dp
00015     REAL(dp), ALLOCATABLE :: remi(:), loss(:)
00016     CHARACTER(LEN=512) :: decay=""
00017     LOGICAL :: detchl, &                                !< flag for Chl in detritus
00018       detpic, &                                         !< flag for PIC in detritus
00019     aggregates=.false.

```



```

00020      INTEGER :: nrec
00021      INTEGER, ALLOCATABLE :: iloss(:), & !< indices of extra loss terms for Chl and O2
00022          isvl(:), & !< state-variable indices corresponding to iloss
00023          irec(:)
00024      PROCEDURE(det_fluxes), POINTER :: fluxes
00025      PROCEDURE(det_loss), POINTER :: losses
00026  CONTAINS
00027      PROCEDURE :: flux => det_flux
00028      PROCEDURE :: read => det_read
00029      PROCEDURE :: set => det_set
00030  END TYPE detritus
00031
00032  CONTAINS
00033
00035  SUBROUTINE det_read (grp, env, lun)
00036      IMPLICIT NONE
00037      CLASS(detritus), TARGET, INTENT(INOUT) :: grp
00038      TYPE(local), TARGET, INTENT(IN) :: env
00039      INTEGER, INTENT(IN) :: lun
00040      CHARACTER(LEN=512) :: fn, decay, ft, msg, species='det'
00041      REAL(dp) :: omax
00042      TYPE(quantity) :: remiC, remiN, remiP, remiPIC, formPIC, remiChl, sink, &
00043          sticky, kag
00044      LOGICAL :: aggregates
00045      INTEGER :: ncon
00046      namelist /det/ remic, remin, remip, remipic, formpic, omax, remichl, sink, decay, &
00047          ft, sticky, kag, species, aggregates
00048      rewind(lun)
00049      DO
00050          remic = quantity(grp%remiC, 'd-1')
00051          remin = quantity(grp%remiN, 'd-1')
00052          remip = quantity(grp%remiP, 'd-1')
00053          remipic = quantity(grp%remiPIC, 'd-1')
00054          formpic = quantity(grp%formPIC, 'd-1')
00055          omax = grp%omax
00056          remichl = quantity(grp%remiChl, 'd-1')
00057          sink = quantity(grp%sink, 'm d-1')
00058          sticky = quantity(grp%sticky, ")
00059          kag = quantity(-1._dp, 'd-1')
00060          aggregates = grp%aggregates
00061          decay = "
00062          ft = 'Eppley'
00063          READ (lun, nml=det, END=100, IOMSG=msg, ERR=900)
00064          IF (trim(species(1:100)).EQ.trim(grp%name)) EXIT
00065      END DO
00066      grp%detPIC = remipic%value.GE.0
00067      grp%detChl = remichl%value.GE.0
00068      grp%decay = decay
00069      ncon = 3 + count((/grp%detPIC, grp%detChl/))
00070      grp%nsv = ncon
00071      grp%nrec = 3 + count((/grp%detPIC/))
00072      ALLOCATE (grp%constituents(ncon))
00073      grp%constituents(1:3) = (/ 'C', 'N', 'P' /)
00074      grp%sticky = sticky%value
00075      IF (aggregates.AND.(kag%value.LT.0._dp)) THEN
00076          WRITE (stderr, '("Missing kag in namelist det for ",A,".")') trim(grp%name)
00077          stop
00078      END IF
00079      grp%aggregates = aggregates
00080      grp%kag = convert(kag, 's-1')
00081      grp%remiC = convert(remic, 's-1')
00082      grp%remiN = convert(remin, 's-1')
00083      grp%remiP = convert(remip, 's-1')
00084      IF (grp%detPIC) THEN
00085          grp%constituents(4) = 'PIC'
00086          grp%remiPIC = max(convert(remipic, 's-1'), 0._dp)
00087          grp%formPIC = convert(formpic, 's-1')
00088          grp%omax = omax
00089      END IF
00090      IF (grp%detChl) THEN
00091          grp%constituents(ncon) = 'Chl'
00092          grp%remiChl = max(convert(remichl, 's-1'), 0._dp)
00093      END IF
00094      grp%sink = convert(sink, 'm s-1')
00095      CALL grp%ft_select (ft, 'det_read')
00096      RETURN
00097 100 CONTINUE
00098      INQUIRE (unit=lun, name=fn)
00099      WRITE (stderr, '("Namelist det missing in file ",A,".")') trim(fn)
00100      stop 1
00101 900 CONTINUE
00102      WRITE (stderr, '("det_read:",A)') trim(msg)
00103      WRITE (stderr, '("searching/reading namelist det for species ",A)') trim(grp%name)
00104      stop 1
00105  END SUBROUTINE det_read
00106
00108  SUBROUTINE det_set (grp, grps, env)

```

```

00109      IMPLICIT NONE
00110      CLASS(detritus), TARGET, INTENT(INOUT) :: grp
00111      TYPE(state), DIMENSION(:), INTENT(IN) :: grps
00112      TYPE(local), TARGET, INTENT(INOUT) :: env
00113      INTEGER :: n, igrec, nloss, nxl
00114      LOGICAL rDIC
00115      igrec = 0
00116      ! find index of recipient group (DOM or inorganic matter) of decaying detritus
00117      DO n=1,env%nft
00118         IF (trim(grp%decay).EQ.trim(grps(n)%var%name)) igrec = n
00119      END DO
00120      rdic = igrec.EQ.0
00121      nloss = grp%ncon + count((/rdic/))
00122      nxl = nloss - grp%nrec ! number of extra losses
00123      ALLOCATE (grp%irec(grp%nrec), grp%remi(nloss), grp%loss(nloss), grp%isvl(nloss), grp%iloss(nxl))
00124      grp%remi(1:3) = (/grp%remiC, grp%remiN, grp%remiP/)
00125      grp%isvl(1:grp%nrec) = grp%isv(1:grp%nrec)
00126      IF (rdic) THEN
00127         IF (env%ndom.GT.0) &
00128            WRITE (stderr, '("det_set: DOM present but not identified as decay in namelist det (decay =
",A,")!")') &
00129                trim(grp%decay)
00130         grp%irec(1:3) = (/env%idic, env%idin, env%idip/) ! if no DOM: remineralize directly
00131         grp%iloss(nxl) = nloss
00132         grp%isvl(grp%iloss(nxl)) = env%io2
00133         grp%remi(nloss) = grp%remiC*env%RQ ! remineralisation of POC to DIC consumes O2
00134         env%msk(grp%isv(1),grp%isvl(grp%iloss(nxl)),:) = .true.
00135      ELSE
00136         grp%irec(1:3) = grps(igrec)%var%isv(1:3)
00137      END IF
00138      IF (grp%detPIC) THEN ! PIC becomes DIC (alkalinity changes are accounted for in
plankton:alkalinity)
00139         grp%irec(4) = env%idic
00140         grp%losses => det_loss_pic
00141      ELSE
00142         grp%losses => det_loss
00143      END IF
00144      IF (grp%detChl) THEN ! Chl disappears
00145         grp%iloss(1) = grp%nrec + 1
00146         grp%isvl(grp%iloss(1)) = grp%isv(grp%ncon)
00147         grp%remi(grp%iloss(1)) = grp%remiChl
00148         env%msk(grp%isv(1),grp%isvl(grp%iloss(1)),:) = .true.
00149      END IF
00150      IF (nxl.GT.0) THEN
00151         grp%fluxes => det_fluxes
00152      ELSE
00153         grp%fluxes => det_fluxes_decay
00154      END IF
00155      DO n=1,grp%nrec
00156         env%msk(grp%isv(n),grp%irec(n),:) = .true.
00157      END DO
00158      END SUBROUTINE det_set
00159
00160      SUBROUTINE det_flux (grp, grps, env, box, times)
00161      IMPLICIT NONE
00162      CLASS(detritus), INTENT(INOUT) :: grp
00163      TYPE(state), DIMENSION(:), INTENT(IN) :: grps
00164      TYPE(local), INTENT(INOUT) :: env
00165      TYPE(layer), INTENT(INOUT) :: box(0:)
00166      TYPE(timing), INTENT(IN) :: times
00167      grp%vv = grp%sink
00168      CALL grp%losses (box=box(1))
00169      CALL grp%fluxes (box=box(1))
00170      END SUBROUTINE det_flux
00171
00172      SUBROUTINE det_loss_pic (grp, box)
00173      IMPLICIT NONE
00174      CLASS(detritus), INTENT(INOUT) :: grp
00175      TYPE(layer), INTENT(IN) :: box
00176      grp%remi(4) = grp%remiPIC - max(box%omega - grp%omax, 0._dp)*grp%formPIC
00177      entry det_loss(grp, box)
00178      grp%loss = grp%ft(box)*grp%remi*box%stv(grp%isvl)
00179      END SUBROUTINE det_loss_pic
00180
00181      SUBROUTINE det_fluxes (grp, box)
00182      IMPLICIT NONE
00183      CLASS(detritus), INTENT(IN) :: grp
00184      TYPE(layer), INTENT(INOUT) :: box
00185      INTEGER :: n
00186      box%soma(grp%isv(1),grp%isvl(grp%iloss)) = -grp%loss(grp%iloss)
00187      entry det_fluxes_decay(grp, box)
00188      DO n=1,grp%nrec
00189         box%soma(grp%irec(n),grp%isvl(n)) = -grp%loss(n)
00190         box%soma(grp%isvl(n),grp%irec(n)) = grp%loss(n)
00191      END DO
00192      END SUBROUTINE det_fluxes
00193

```

00194 END MODULE det

7.13 /Users/mpahlow/oppla/src/dic.f90 File Reference

Data Types

- type `dic::dicsys`

Modules

- module `dic`
Functions for the carbonate system and alkalinity.

Functions/Subroutines

- elemental real(dp) function `asf_o2` (box)
- subroutine `dic::asfco2` (dic, env, box)
Air-to-sea flux of CO₂ and O₂.
- subroutine `dic::co2` (dic, box, niter)
Concentrations of protons and DIC species and aragonite and calcite saturation.
- real(dp) function `deltaalk` (dic, hplus)
- subroutine `dic::dic_potalk` (dic, box)
alkalinity as a function of potential alkalinity and DIN
- subroutine `dic::dicalp` (dic, box)
parameters for the DIC system and alkalinity (total scale)
- real(dp) elemental function `dic::eos80_rho` (tmp, sal)
seawater density in g m⁻³ after [Millero et al. \(1980\)](#)
- elemental real(dp) function `dic::faco2` (tk, prat, xco2)
Fugacity of CO₂ in air.
- subroutine `dic::hini_f06` (dic, box)
Initialise DIC system after [Follows et al. \(2006\)](#).
- subroutine `dic::hini_m13` (dic, box)
Initial guess for [H+] after [Munhoven \(2013\)](#).
- subroutine `hset_m13` (dic, box)
- subroutine `dic::hsolve_f06` (dic, box, niter)
Concentrations of H⁺ from alkalinity and total DIC concentration [H⁺] is calculated after [Follows et al. \(2006\)](#) but including sulphate and fluoride systems.
- subroutine `dic::hsolve_m13` (dic, box, niter)
Solve for [H+] given alkalinity and total DIC concentration.
- subroutine `dic::indic` (dics, lun, env, fluxes)
Initialise DIC module.
- elemental real(dp) function `dic::k0co2` (box)
CO₂ solubility
- elemental real(dp) function `dic::lgkspa` (sal, tk, tc, prs)
Equilibrium constant for aragonite after [Millero \(1995\)](#).
- elemental real(dp) function `dic::lgkspc` (sal, tk, tc, prs)
Equilibrium constant for calcite after [Millero \(1995\)](#).
- elemental real(dp) function `dic::lnk1_sws` (sal, tk, lntk, tc, prs, prt, ds, sqs)

- $k1 = [H^+][HCO_3^-]/[CO_2]$: equilibrium constant for CO_2 and HCO_3^- in $mol\,kg^{-1}$ for SWS
- elemental real(dp) function [dic::lnk1_total](#) (sal, tk, lntk, tc, prs, prt, ds)
- $k1 = [H^+][HCO_3^-]/[CO_2]$: equilibrium constant for CO_2 and HCO_3^- in $mol\,kg^{-1}$ for the total scale
- elemental real(dp) function [dic::lnk1p_sws](#) (sal, tk, lntk, tc, prs, prt, sqs)
- First equilibrium constant of the phosphate system.
- elemental real(dp) function [dic::lnk2_sws](#) (sal, tk, lntk, tc, prs, prt, ds, sqs)
- $k2 = [H^+][CO_3^{2-}]/[HCO_3^-]$: equilibrium constant for HCO_3^- and CO_3^{2-} in $mol\,kg^{-1}$ for SWS
- elemental real(dp) function [dic::lnk2_total](#) (sal, tk, lntk, tc, prs, prt, ds)
- $k2 = [H^+][CO_3^{2-}]/[HCO_3^-]$: equilibrium constant for HCO_3^- and CO_3^{2-} in $mol\,kg^{-1}$ for the total scale
- elemental real(dp) function [dic::lnk2p_sws](#) (sal, tk, lntk, tc, prs, prt, sqs)
- Second equilibrium constant of the phosphate system.
- elemental real(dp) function [dic::lnk3p_sws](#) (sal, tk, tc, prs, prt, sqs)
- Third equilibrium constant of the phosphate system.
- elemental real(dp) function [dic::lnkb_total](#) (sal, tk, lntk, tc, prs, prt, ds, sqs)
- Boron system.
- elemental real(dp) function [dic::lnkf_free](#) (tk, tc, prs, prt, i0)
- Flouride system (free scale)
- elemental real(dp) function [dic::lnkf_total](#) (tk, tc, prs, prt, sqs)
- Flouride system (total scale)
- elemental real(dp) function [dic::lnknh4_sws](#) (sal, tk, tc, prs, prt, sqs)
- Ammonium system (SWS)
- elemental real(dp) function [dic::lnks_free](#) (tk, lntk, tc, prs, prt, i0)
- Sulfate system.
- elemental real(dp) function [dic::lnkw_sws](#) (sal, tk, lntk, tc, prs, prt, sqs)
- Carbonate SYSTEM.
- subroutine [dic::pivel_lm86](#) (dic, env, box)
- Piston velocity after [Liss and Merlivat \(1986\)](#).
- subroutine [dic::pivel_w14](#) (dic, env, box)
- Piston velocity after [Wanninkhof \(2014\)](#).
- subroutine [dic::pivel_w92](#) (dic, env, box)
- Piston velocity after [Wanninkhof \(1992\)](#).
- subroutine [dic::pivel_w99](#) (dic, env, box)
- Piston velocity after [Wanninkhof and McGillis \(1999\)](#).

Variables

- real(dp), parameter [dic::rgas](#) = 83.1446261815324_dp
ideal gas constant in $dPa\,m^3\,K^{-1}\,mol^{-1}$ (this value is from wikipedia, citing 2018 CODAT, NIST)
- real(dp), parameter [dic::t0ck](#) = 273.15_dp
0 °C in K

7.13.1 Function/Subroutine Documentation

7.13.1.1 asf_o2()

```
elemental real(dp) function asfco2::asf_o2 (
    type(layer), intent(in) box ) [private]
```

Definition at line 204 of file [dic.f90](#).

References [dic::t0ck](#).

Referenced by [dic::asfco2\(\)](#).

Here is the caller graph for this function:



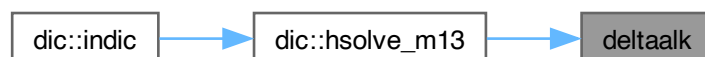
7.13.1.2 `deltaalk()`

```
real(dp) function hsolve_m13::deltaalk (  
    class(dicsys), intent(in) dic,  
    real(dp), intent(in) hplus ) [private]
```

Definition at line [878](#) of file [dic.f90](#).

Referenced by [dic::hsolve_m13\(\)](#).

Here is the caller graph for this function:



7.13.1.3 `hset_m13()`

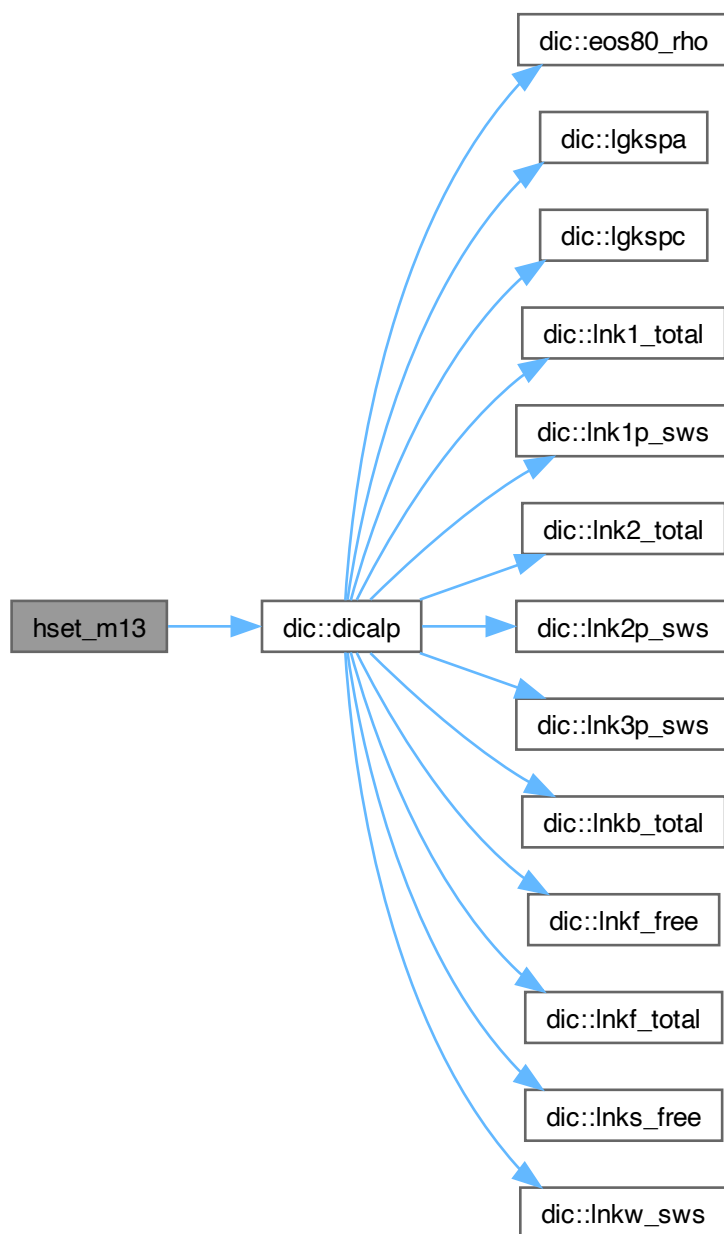
```
subroutine hini_m13::hset_m13 (  
    class(dicsys), intent(inout) dic,  
    type(layer), intent(inout) box ) [private]
```

Definition at line [757](#) of file [dic.f90](#).

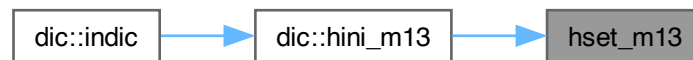
References [dic::dicalp\(\)](#).

Referenced by [dic::hini_m13\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.14 dic.f90

[Go to the documentation of this file.](#)

```

00001
00061
00062 MODULE dic
00063   USE et
00064   IMPLICIT NONE
00065   PRIVATE
00066   REAL(dp), PARAMETER :: t0ck=273.15_dp
00067   REAL(dp), PARAMETER :: rgas = 83.1446261815324_dp
00069   TYPE, PUBLIC :: dicsys
00070     REAL(dp) :: at, pt, tk, ph0=8.1_dp, ca=10.28e-3_dp,&
00071     kw, k1, k12, k2, kb, kf, k12p, ksi, kso4, klp, k2p, k3p, knh4,&
00072     kspc, kspa, so4, bt, ct, ft, sit, ac, gmk, acd,&
00073     rrnsi,&
00074     rrrnp,&
00075     potalk=0._dp,&
00076     totalk=0._dp,&
00077     r_tot_free,&
00078     r_tot_sws,&
00079     faco2, k0co2,&
00080     schnco2,&
00081     schno2,&
00082     tvco2,&
00083     tvo2
00084     REAL(dp), POINTER :: asfco2, asfo2
00085     INTEGER :: niter=2
00086     PROCEDURE(dic_alk), POINTER :: alkalinity
00087     PROCEDURE(pivel_lm86), POINTER :: pivel
00088     PROCEDURE(hsolve_f06), POINTER :: hsolve
00089     PROCEDURE(hini_f06), POINTER :: init
00090   CONTAINS
00091     PROCEDURE :: read => indic
00092     PROCEDURE :: asf => asfco2
00093     PROCEDURE :: co2 => co2
00094   END TYPE dicsys
00095   CONTAINS
00096
00109   ! Values from namelist envi (in plankton_init) values for potalk, co2, and atmprs are used only if
00110   ! no corresponding datasets are found in the forcing file
00111   SUBROUTINE indic (dics, lun, env, fluxes)
00112     IMPLICIT NONE
00113     CLASS(dicsys), INTENT(INOUT) :: dics
00114     INTEGER, INTENT(IN) :: lun
00115     TYPE(local), INTENT(INOUT) :: env
00116     REAL(dp), DIMENSION(:,0:), TARGET, INTENT(IN), OPTIONAL :: fluxes
00117     REAL(dp) :: pH
00118     CHARACTER(LEN=15) :: pvfun='LM86', phfun='Munhoven 2013'
00119     CHARACTER(LEN=512) :: iomsg
00120     INTEGER :: niter
00121     TYPE(quantity) :: rfrnsi, rfrnp
00122     namelist /dic/ ph, rfrnsi, rfrnp, pvfun, phfun, niter
00123     IF (PRESENT(fluxes)) THEN
00124       dics%asfco2 => fluxes(env%idic,0)
00125       dics%asfo2 => fluxes(env%io2,0)
00126     ELSE
00127       ALLOCATE (dics%asfco2, dics%asfo2)
00128       dics%asfco2 = 0._dp
00129       dics%asfo2 = 0._dp
00130     END IF
00131     ph = dics%ph0
00132     niter = dics%niter
00133     rfrnsi = quantity(1d0, 'molN/molSi') ! N:Si Redfield ratio

```

```

00134     rfrnp = quantity(16d0, 'molN/molP') ! N:P Redfield ratio
00135     rewind(lun)
00136     READ (lun,nml=dic,END=10,IOMSG=iomsg,ERR=900)
00137 10 CONTINUE
00138     dics%niter = niter
00139     SELECT CASE (trim(pvfun))
00140     CASE ('LM86') ! Liss & Merlivat (1986)
00141         dics%pivel => pivel_lm86
00142     CASE ('W92') ! Wanninkhof (1992)
00143         dics%pivel => pivel_w92
00144     CASE ('WG99') ! Wanninkhof and McGillis (1999)
00145         dics%pivel => pivel_w99
00146     CASE ('W14') ! Wanninkhof (2014)
00147         dics%pivel => pivel_w14
00148     CASE DEFAULT
00149         stop 'indic: Unknown piston velocity function.'
00150     END SELECT
00151     SELECT CASE (trim(phfun))
00152     CASE ('Follows 2006', 'F06', 'Follows06')
00153         dics%init => hini_f06
00154         dics%Hsolve => hsolve_f06
00155         dics%pH0 = ph
00156     CASE ('Munhoven 2013', 'M13', 'Munhoven13')
00157         dics%init => hini_m13
00158         dics%Hsolve => hsolve_m13
00159     CASE DEFAULT
00160         stop 'Invalid pHfun.'
00161     END SELECT
00162     dics%rrnsi = convert(rfrnsi, 'molN/molSi')
00163     dics%rrnp = convert(rfrnp, 'molN/molP')
00164     IF ((env%ialk.EQ.0).AND.(dics%potalk.GT.0d0)) THEN
00165         IF (dics%totalk.GT.0d0) stop &
00166             'indic: Specify only one of potalk or totalk in namelist envi'
00167         dics%alkalinity => dic_potalk
00168     ELSE
00169         dics%alkalinity => dic_alk
00170     END IF
00171     RETURN
00172 900 CONTINUE
00173     WRITE (stderr,('("indic:",A)') trim(iomsg)
00174     stop
00175 END SUBROUTINE indic
00176
00192 SUBROUTINE asfco2 (dic, env, box)
00193     IMPLICIT NONE
00194     CLASS(dicsys), INTENT(INOUT) :: dic
00195     TYPE(local), INTENT(IN) :: env
00196     TYPE(layer), INTENT(INOUT) :: box
00197     CALL dic%pivel (env=env, box=box) ! piston velocity
00198     dic%faco2 = faco2(box%tk, env%pressure, env%xco2)
00199     dic%k0co2 = k0co2(box)
00200     dic%asfco2 = dic%tvco2*(dic%k0co2*dic%faco2 - box%CO2)*1d6 & ! convert m*mol/kg/s to mmol/m2/s
00201         + env%sfDIC
00202     dic%asfo2 = asf_o2(box) + env%sfO2
00203 CONTAINS
00204 ELEMENTAL FUNCTION asf_o2 (box) RESULT (asfo2)
00205     IMPLICIT NONE
00206     REAL(dp), PARAMETER :: a0=5.80871_dp, a1=3.20291_dp, a2=4.17887_dp,&
00207         a3=5.10006_dp, a4=-9.86643e-2_dp, a5=3.80369_dp, b0=-7.01577e-3_dp,&
00208         b1=-7.70028e-3_dp, b2=-1.13864e-2_dp, b3=-9.51519e-3_dp, c0=-2.75915e-7_dp
00209     TYPE(layer), INTENT(IN) :: box
00210     REAL(dp) :: asfo2, ts, o2sat
00211     ts = log((298.15d0 - box%temperature)/(t0ck + box%temperature))
00212     ! O2 saturation concentration [mol/kg=mmol/m3]
00213     o2sat = exp(a0 + ts*(a1 + ts*(a2 + ts*(a3 + ts*(a4 + ts*a5)))) &
00214         + box%salinity*(b0 + ts*(b1 + ts*(b2 + ts*b3)) + box%salinity*c0))
00215     asfo2 = dic%tvo2*(o2sat - box%o2)
00216 END FUNCTION asf_o2
00217 END SUBROUTINE asfco2
00218
00224 SUBROUTINE pivel_lm86 (dic, env, box)
00225     IMPLICIT NONE
00226     CLASS(dicsys), INTENT(INOUT) :: dic
00227     TYPE(local), INTENT(IN) :: env
00228     TYPE(layer), INTENT(IN) :: box
00229     IF (env%wind.LE.3.6d0) THEN
00230         dic%tvco2 = 0.17d0*env%wind/3.6d5
00231     ELSEIF (env%wind.LE.13d0) THEN
00232         dic%tvco2 = (2.85d0*env%wind - 9.65d0)/3.6d5
00233     ELSE
00234         dic%tvco2 = (5.9d0*env%wind - 49.3d0)/3.6d5
00235     END IF
00236     dic%tvo2 = dic%tvco2
00237 END SUBROUTINE pivel_lm86
00238
00247 SUBROUTINE pivel_w92 (dic, env, box)
00248     IMPLICIT NONE

```



```

00249 CLASS(dicsys), INTENT(INOUT) :: dic
00250 TYPE(local), INTENT(IN) :: env
00251 TYPE(layer), INTENT(IN) :: box
00252 REAL(dp) :: fwind
00253 ! Schmidt number for CO2 after Wanninkhof (1992)
00254 ! Schmidt number for O2 from UVic 2.9
00255 dic%schnco2 = 2073.1_dp - box%temperature*(125.62_dp &
00256 - box%temperature*(3.6276_dp - box%temperature*0.043219_dp))
00257 dic%schno2 = 1638._dp - box%temperature*(81.83_dp &
00258 - box%temperature*(1.483_dp - box%temperature*8.004e-3_dp))
00259 fwind = 0.31_dp*env%wind**2/3.6e5_dp ! convert cm/h to m/s
00260 dic%tvco2 = fwind/sqrt(dic%schnco2/660._dp)
00261 dic%tvo2 = fwind/sqrt(dic%schno2/660._dp)
00262 END SUBROUTINE pivel_w92
00263
00272 SUBROUTINE pivel_w99 (dic, env, box)
00273 IMPLICIT NONE
00274 CLASS(dicsys), INTENT(INOUT) :: dic
00275 TYPE(local), INTENT(IN) :: env
00276 TYPE(layer), INTENT(IN) :: box
00277 REAL(dp) :: fwind
00278 ! Schmidt number for CO2 after Wanninkhof (1992)
00279 ! Schmidt number for O2 from UVic 2.9
00280 dic%schnco2 = 2073.1_dp - box%temperature*(125.62_dp &
00281 - box%temperature*(3.6276_dp - box%temperature*0.043219_dp))
00282 dic%schno2 = 1638._dp - box%temperature*(81.83_dp &
00283 - box%temperature*(1.483_dp - box%temperature*8.004e-3_dp))
00284 fwind = 0.0283_dp*env%wind**3/3.6e5_dp ! convert cm/h to m/s
00285 dic%tvco2 = fwind/sqrt(dic%schnco2/660._dp)
00286 dic%tvo2 = fwind/sqrt(dic%schno2/660._dp)
00287 END SUBROUTINE pivel_w99
00288
00297 SUBROUTINE pivel_w14 (dic, env, box)
00298 IMPLICIT NONE
00299 CLASS(dicsys), INTENT(INOUT) :: dic
00300 TYPE(local), INTENT(IN) :: env
00301 TYPE(layer), INTENT(IN) :: box
00302 REAL(dp) :: fwind
00303 ! Schmidt numbers for CO2 and O2 after Wanninkhof (2014)
00304 dic%schnco2 = 2116.8_dp - box%temperature*(136.25_dp - box%temperature*(4.7353_dp &
00305 - box%temperature*(9.2307e-2_dp - box%temperature*7.555e-4_dp)))
00306 dic%schno2 = 1920.4_dp - box%temperature*(135.6_dp - box%temperature*(5.2122_dp &
00307 - box%temperature*(0.10939_dp - box%temperature*9.3777e-4_dp)))
00308 fwind = 0.251_dp*env%wind**2/3.6e5_dp ! convert cm/h to m/s
00309 dic%tvco2 = fwind/sqrt(dic%schnco2/660._dp)
00310 dic%tvo2 = fwind/sqrt(dic%schno2/660._dp)
00311 END SUBROUTINE pivel_w14
00312
00319 ELEMENTAL FUNCTION k0co2 (box) RESULT (k0)
00320 IMPLICIT NONE
00321 REAL(dp), PARAMETER :: atm=101325._dp ! 1 atm in Pa
00322 TYPE(layer), INTENT(IN) :: box
00323 REAL(dp) :: lnk0, k0
00324 lnk0 = 9345.17_dp/box%tk - 167.81077_dp + 23.3585_dp*log(box%tk) & ! (7.1.3)
00325 + box%salinity*(2.3517e-2_dp - box%tk*(2.3656e-4_dp - box%tk*4.7036e-7_dp))
00326 k0 = exp(lnk0)/atm
00327 END FUNCTION k0co2
00328
00337 ELEMENTAL FUNCTION faco2 (tk, prat, xco2) RESULT (fugacity)
00338 IMPLICIT NONE
00339 REAL(dp), INTENT(IN) :: tk
00340 REAL(dp), INTENT(IN) :: prat
00341 REAL(dp), INTENT(IN) :: xco2
00342 REAL(dp) :: vcco2,& ! virial coefficient of pure CO2 in m3/mol
00343 vcairco2,& ! virial coefficient of CO2 in air in m3/mol
00344 fugacity
00345 vcco2 = -1636.75e-6_dp &
00346 + tk*(12.0408e-6_dp - tk*(3.27957e-8_dp - tk*3.16528e-11_dp)) ! SOP 24 (19)
00347 vcairco2 = 57.7e-6_dp - 0.118e-6_dp*tk ! SOP 24 (20)
00348 fugacity = xco2*prat*exp((vcco2 + 2._dp*(1._dp - xco2)**2*vcairco2)&
00349 *prat*10._dp/rgas/tk) ! SOP 24 (15)
00350 END FUNCTION faco2
00351
00356 ELEMENTAL FUNCTION lnkw_sws (sal, tk, lntk, tc, prs, prt, sqs) RESULT (lnkw)
00357 IMPLICIT NONE
00358 REAL(dp), INTENT(IN) :: sal, tk, lntk, tc, prs, prt, sqs
00359 REAL(dp) :: lnkw, dvi, dki
00360 dvi = -20.02_dp + (0.1119_dp - 0.1409e-02_dp*tc)*tc
00361 dki = -5.13e-3_dp + 0.0794e-3_dp*tc
00362 lnkw = -13847.26_dp/tk + 148.9802_dp - 23.6521_dp*lntk&
00363 + (118.67_dp/tk - 5.977_dp + 1.0495_dp*lntk)*sqs - 0.01615_dp*sal &
00364 + (-dvi + 0.5_dp*dki*prs)*prt ! pressure correction
00365 END FUNCTION lnkw_sws
00366
00370 ELEMENTAL FUNCTION lnk1_sws (sal, tk, lntk, tc, prs, prt, ds, sqs) RESULT (lnk1)
00371 IMPLICIT NONE
00372 REAL(dp), INTENT(IN) :: sal, tk, lntk, tc, prs, prt, ds, sqs

```

```

00373     REAL(dp) :: lnk1, dvi, dki
00374     dvi = -25.5_dp - 0.151_dp*ds + 0.1271_dp*tc
00375     dki = -3.08e-3_dp - 0.578e-3_dp*ds + 0.0877e-3_dp*tc
00376     lnk1 = -2275.036_dp/tk + 2.18867_dp - 1.468591_dp*lnk &
00377           + (-9.33291_dp/tk - 0.138681_dp - 0.00574938_dp*sal)*sqs + 0.0726483_dp*sal &
00378           + (-dvi + 0.5_dp*dki*prs)*prt
00379 END FUNCTION lnk1_sws
00380
00384 ELEMENTAL FUNCTION lnk2_sws (sal, tk, lntk, tc, prs, prt, ds, sqs) RESULT (lnk2)
00385     IMPLICIT NONE
00386     REAL(dp), INTENT(IN) :: sal, tk, lntk, tc, prs, prt, ds, sqs
00387     REAL(dp) :: lnk2, dvi, dki
00388     dvi = -15.82_dp + 0.321_dp*ds - 0.0219_dp*tc
00389     dki = 1.13e-3_dp - 0.314e-3_dp*ds - 0.1475e-3_dp*tc
00390     lnk2 = -3741.1288_dp/tk - 0.84226_dp - 1.437139_dp*lnk &
00391           + (-24.41239_dp/tk - 0.128417_dp - 0.0091284_dp*sal)*sqs &
00392           + 0.1195308_dp*sal + (-dvi + 0.5_dp*dki*prs)*prt
00393 END FUNCTION lnk2_sws
00394
00398 ELEMENTAL FUNCTION lnk1_total (sal, tk, lntk, tc, prs, prt, ds) RESULT (lnk1)
00399     IMPLICIT NONE
00400     REAL(dp), INTENT(IN) :: sal, tk, lntk, tc, prs, prt, ds
00401     REAL(dp) :: lnk1, dvi, dki
00402     dvi = -25.5_dp - 0.151_dp*ds + 0.1271_dp*tc
00403     dki = -3.08e-3_dp - 0.578e-3_dp*ds + 0.0877e-3_dp*tc
00404     lnk1 = (-3633.86_dp/tk + 61.2172_dp - 9.6777_dp*lnk &
00405           + (0.011555_dp - 1.152e-4_dp*sal)*sal)*log(10._dp) &
00406           + (-dvi + 0.5_dp*dki*prs)*prt
00407 END FUNCTION lnk1_total
00408
00412 ELEMENTAL FUNCTION lnk2_total (sal, tk, lntk, tc, prs, prt, ds) RESULT (lnk2)
00413     IMPLICIT NONE
00414     REAL(dp), INTENT(IN) :: sal, tk, lntk, tc, prs, prt, ds
00415     REAL(dp) :: lnk2, dvi, dki
00416     dvi = -15.82_dp + 0.321_dp*ds - 0.0219_dp*tc
00417     dki = 1.13e-3_dp - 0.314e-3_dp*ds - 0.1475e-3_dp*tc
00418     lnk2 = (-471.78_dp/tk - 25.929_dp + 3.16967_dp*lnk &
00419           + (0.01781_dp - 1.122e-4_dp*sal)*sal)*log(10._dp) &
00420           + (-dvi + 0.5_dp*dki*prs)*prt
00421 END FUNCTION lnk2_total
00422
00429 ELEMENTAL FUNCTION lnks_free (tk, lntk, tc, prs, prt, i0) RESULT (lnks)
00430     IMPLICIT NONE
00431     REAL(dp), INTENT(IN) :: tk, lntk, tc, prs, prt, i0
00432     REAL(dp) :: lnks, dvi, dki
00433     dvi = -18.03_dp + tc*(0.0466_dp + tc*0.316e-03_dp)
00434     dki = -4.53e-3_dp + tc*0.09e-3_dp
00435     lnks = -4276.1_dp/tk + 141.328_dp - 23.093_dp*lnk &
00436           + (-13856._dp/tk + 324.57_dp - 47.986_dp*lnk)*sqrt(i0) &
00437           + (35474._dp/tk - 771.54_dp + 114.723_dp*lnk)*i0 &
00438           + (-2698._dp*sqrt(i0) + 1776._dp*i0)*i0/tk &
00439           + (-dvi + 0.5_dp*dki*prs)*prt ! pressure correction
00440 END FUNCTION lnks_free
00441
00447 ELEMENTAL FUNCTION lnkb_total (sal, tk, lntk, tc, prs, prt, ds, sqs) RESULT (lnkb)
00448     IMPLICIT NONE
00449     REAL(dp), INTENT(IN) :: sal, tk, lntk, tc, prs, prt, ds, sqs
00450     REAL(dp) :: lnkb, dvi, dki
00451     dvi = -29.48_dp + 0.295_dp*ds + (0.1622_dp - 0.002608_dp*tc)*tc
00452     dki = -2.84e-3_dp + 0.354e-3_dp*ds
00453     lnkb = (-8966.9_dp - 2890.53_dp*sqs - (77.942_dp - 1.728_dp*sqs + 0.0996*sal)*sal)/tk &
00454           + 148.0248_dp + (137.1942_dp + 0.053105_dp*tk)*sqs + 1.62142_dp*sal &
00455           + (-24.4344_dp - 25.085_dp*sqs - 0.2474_dp*sal)*lntk &
00456           + (-dvi + 0.5_dp*dki*prs)*prt ! pressure correction
00457 END FUNCTION lnkb_total
00458
00464 ELEMENTAL FUNCTION lnknh4_sws (sal, tk, tc, prs, prt, sqs) RESULT (lnknh4)
00465     IMPLICIT NONE
00466     REAL(dp), INTENT(IN) :: sal, tk, tc, prs, prt, sqs
00467     REAL(dp) :: lnknh4, dvi, dki
00468     dvi = -26.43_dp + tc*(0.0889_dp - tc*0.905e-03_dp) ! volume change for ionization
00469     dki = -5.03e-3_dp + tc*0.0814e-3_dp ! dki compressibility change for ionization
00470     lnknh4 = -0.25444_dp - 6285.33_dp/tk + 0.0001635_dp*tk &
00471           + (0.46532_dp - 123.7184_dp/tk)*sqs + (-0.01992_dp + 3.17556_dp/tk)*sal &
00472           + (-dvi + 0.5_dp*dki*prs)*prt ! pressure correction
00473 END FUNCTION lnknh4_sws
00474
00480 ELEMENTAL FUNCTION lnkf_free (tk, tc, prs, prt, i0) RESULT (lnkf)
00481     IMPLICIT NONE
00482     REAL(dp), INTENT(IN) :: tk, tc, prs, prt, i0
00483     REAL(dp) :: lnkf, dvi, dki
00484     dvi = -9.78_dp + tc*(-0.009_dp - tc*0.942e-03_dp) ! volume change for ionization
00485     dki = -3.91e-3_dp + tc*0.054e-3_dp ! dki compressibility change for ionization
00486     lnkf = 1590.2_dp/tk - 12.641_dp + 1.525_dp*sqrt(i0) &
00487           + (-dvi + 0.5_dp*dki*prs)*prt ! pressure correction
00488 END FUNCTION lnkf_free
00489

```

```

00495 ELEMENTAL FUNCTION lnkf_total (tk, tc, prs, prt, sqs) RESULT (lnkf)
00496 IMPLICIT NONE
00497 REAL(dp), INTENT(IN) :: tk, tc, prs, prt, sqs
00498 REAL(dp) :: lnkf, dvi, dki
00499 dvi = -9.78_dp + tc*(-0.009_dp - tc*0.942e-03_dp) ! volume change for ionization
00500 dki = -3.91e-3_dp + tc*0.054e-3_dp ! dki compressibility change for ionization
00501 lnkf = 874._dp/tk - 9.68_dp + 0.111_dp*sqs &
00502 + (-dvi + 0.5_dp*dki*prs)*prt ! pressure correction
00503 END FUNCTION lnkf_total
00504
00505 ! equilibrium constants of phosphate system in mol/kg
00506 ! k1p = [H+][H2PO4-]/[H3PO4]
00507 ! k2p = [H+][HPO4^2-]/[H2PO4-]
00508 ! k3p = [H+][PO4^3-]/[HPO4^2-]
00509
00511 ELEMENTAL FUNCTION lnk1p_sws (sal, tk, lntk, tc, prs, prt, sqs) RESULT (lnk1p)
00512 IMPLICIT NONE
00513 REAL(dp), INTENT(IN) :: sal, tk, lntk, tc, prs, prt, sqs
00514 REAL(dp) :: lnk1p, dvi, dki
00515 dvi = -14.51_dp + (0.1211_dp - 0.321e-03_dp*tc)*tc ! volume change for ionization
00516 dki = -2.67e-3_dp + 0.0427e-3_dp*tc ! compressibility change for ionization
00517 lnk1p = -4576.752_dp/tk + 115.54_dp - 18.453_dp*lntk &
00518 + (-106.736_dp/tk + 0.69171_dp)*sqs + (-0.65643_dp/tk - 1.844e-2_dp)*sal &
00519 + (-dvi + 0.5_dp*dki*prs)*prt ! pressure correction
00520 END FUNCTION lnk1p_sws
00521
00525 ELEMENTAL FUNCTION lnk2p_sws (sal, tk, lntk, tc, prs, prt, sqs) RESULT (lnk2p)
00526 IMPLICIT NONE
00527 REAL(dp), INTENT(IN) :: sal, tk, lntk, tc, prs, prt, sqs
00528 REAL(dp) :: lnk2p, dvi, dki
00529 dvi = -23.12_dp + (0.1758_dp - 2.647e-03_dp*tc)*tc ! volume change for ionization
00530 dki = -5.15e-3_dp + 0.09e-3_dp*tc ! compressibility change for ionization
00531 lnk2p = -8814.715_dp/tk + 172.1033_dp - 27.927_dp*lntk &
00532 + (-160.34_dp/tk + 1.3566_dp)*sqs + (0.37335_dp/tk - 5.778d-2_dp)*sal &
00533 + (-dvi + 0.5_dp*dki*prs)*prt ! pressure correction
00534 END FUNCTION lnk2p_sws
00535
00540 ELEMENTAL FUNCTION lnk3p_sws (sal, tk, tc, prs, prt, sqs) RESULT (lnk3p)
00541 IMPLICIT NONE
00542 REAL(dp), INTENT(IN) :: sal, tk, tc, prs, prt, sqs
00543 REAL(dp) :: lnk3p, dvi, dki
00544 dvi = -26.57_dp + (0.202_dp - 3.042e-03_dp*tc)*tc ! volume change for ionization
00545 dki = -4.08e-3_dp + 0.0714e-3_dp*tc ! compressibility change for ionization
00546 lnk3p = -3070.75_dp/tk - 18.126_dp + (17.27039_dp/tk + 2.81197_dp)*sqs &
00547 + (-44.99486_dp/tk - 9.984e-2_dp)*sal &
00548 + (-dvi + 0.5_dp*dki*prs)*prt ! pressure correction
00549 END FUNCTION lnk3p_sws
00550
00558 ELEMENTAL FUNCTION lgkspa (sal, tk, tc, prs) RESULT (lgk)
00559 IMPLICIT NONE
00560 REAL(dp), INTENT(IN) :: sal, tk, tc, prs
00561 REAL(dp) :: dv, dk, lgk
00562 dv = -35._dp + 0.5304_dp*tc ! Eq. (91)
00563 dk = -11.76e-3_dp + 0.3692e-3_dp*tc ! Eq. (92)
00564 lgk = -171.945_dp - 0.077993_dp*tk + 2903.293_dp/tk + 71.595_dp*log10(tk) & ! Eq. (79)
00565 + (-0.068393_dp + 1.7276e-3_dp*tk + 88.135_dp/tk + 5.9415e-3_dp*sal)*sqrt(sal) -
00566 0.10018_dp*sal &
00567 - (dv - 0.5_dp*dk*prs)/(rgas*tk)*prs ! Eq. (90)
00568 END FUNCTION lgkspa
00569
00576 ELEMENTAL FUNCTION lgkspc (sal, tk, tc, prs) RESULT (lgk)
00577 IMPLICIT NONE
00578 REAL(dp), INTENT(IN) :: sal, tk, tc, prs
00579 REAL(dp) :: dv, dk, lgk
00580 dv = -48.76_dp + 0.5304_dp*tc ! Eq. (91)
00581 dk = -11.76e-3_dp + 0.3692e-3_dp*tc ! Eq. (92)
00582 lgk = -171.9065_dp - 0.077993_dp*tk + 2839.319_dp/tk + 71.595_dp*log10(tk) & ! Eq. (78)
00583 + (-0.07712_dp + 2.8426e-3_dp*tk + 178.34_dp/tk + 4.1249e-3_dp*sal)*sqrt(sal) -
00584 0.07711_dp*sal &
00585 - (dv - 0.5_dp*dk*prs)/(rgas*tk)*prs ! Eq. (90)
00586 END FUNCTION lgkspc
00587
00596 SUBROUTINE dicalp (dic, box)
00597 IMPLICIT NONE
00598 CLASS(dicsys), INTENT(INOUT) :: dic
00599 TYPE(layer), INTENT(INOUT) :: box
00600 REAL(dp), PARAMETER :: rcf=68e-6_dp/35._dp, rcb=0.232d-3, rcso4=0.14_dp/96.062_dp/1.80655_dp
00601 REAL(dp) :: i0, lntk, lnksi, sqs, mH2O, kS04_free, prs, prt, delsal
00602 prs = 0.1_dp*box%depth ! pressure in bar
00603 delsal = box%salinity - 34.8_dp
00604 sqs = sqrt(box%salinity)
00605 mh2o = 1._dp - 1.005e-3_dp*box%salinity ! mass of H2O in 1 kg seawater
00606 i0 = 19.924e-3_dp*box%salinity/mh2o ! ionic strength
00607 box%tk = box%temperature + t0ck ! convert °C to K
00608 prt = prs/(rgas*box%tk) ! (pressure in bar)/(R*T)
00609 lntk = log(box%tk)
00610 box%rho = eos80_rho(tmp=box%temperature, sal=box%salinity)

```

```

00611     dic%at = box%alkalinity/box%rho ! convert mmol/m3 to mol/kg
00612     dic%ct = box%DIC/box%rho
00613     dic%pt = box%DIP/box%rho
00614     dic%sit = box%silc/box%rho
00615     dic%ft = rcf*box%salinity
00616     dic%bt = rcb/10.811_dp*box%salinity/1.80655_dp
00617     kso4_free = exp(lnks_free(tk=box%tk, lntk=lntk, tc=box%temperature, prs=prs, prt=prt, i0=i0))*mh2o
00618     dic%SO4 = rcso4*box%salinity
00619     dic%r_tot_free = 1._dp + dic%SO4/kso4_free
00620     dic%r_tot_sws = dic%r_tot_free/(dic%ft/exp(lnkf_free(tk=box%tk, tc=box%temperature, prs=prs, &
00621     prt =prt, i0=i0)) + dic%r_tot_free)
00622     ! water
00623     dic%kw = exp(lnkw_sws(sal=box%salinity, tk=box%tk, lntk=lntk, tc=box%temperature, &
00624     prs=prs, prt=prt, sqs=sqs))*dic%r_tot_sws
00625     ! carbon system
00626     dic%k1 = exp(lnk1_total(sal=box%salinity, tk=box%tk, lntk=lntk, tc=box%temperature, &
00627     prs=prs, prt=prt, ds=delsal))
00628     dic%k2 = exp(lnk2_total(sal=box%salinity, tk=box%tk, lntk=lntk, tc=box%temperature, &
00629     prs=prs, prt=prt, ds=delsal))
00630     dic%k12 = dic%k1*dic%k2
00631     ! boron buffer
00632     dic%kb = exp(lnkb_total(sal=box%salinity, tk=box%tk, lntk=lntk, tc=box%temperature, &
00633     prs=prs, prt=prt, ds=delsal, sqs=sqs))
00634     ! sulphate buffer
00635     dic%kSO4 = kso4_free*dic%r_tot_free
00636     ! flouride
00637     dic%kf = exp(lnkf_total(tk=box%tk, tc=box%temperature, prs=prs, prt=prt, sqs=sqs))
00638     ! silicic acid
00639     lnksi = -8904.2_dp/box%tk + 117.4_dp - 19.334_dp*lntk&
00640     + (-458.79_dp/box%tk + 3.5913_dp)*sqrt(i0) + ((188.74_dp/box%tk - 1.5998_dp) &
00641     + (-12.1652_dp/box%tk + 7.871e-2_dp)*i0)*i0
00642     dic%ksi = exp(lnksi)*mh2o*dic%r_tot_sws
00643     ! phosphate buffer
00644     dic%k1p = exp(lnk1p_sws(sal=box%salinity, tk=box%tk, lntk=lntk, tc=box%temperature, &
00645     prs=prs, prt=prt, sqs=sqs))*dic%r_tot_sws
00646     dic%k2p = exp(lnk2p_sws(sal=box%salinity, tk=box%tk, lntk=lntk, tc=box%temperature, &
00647     prs=prs, prt=prt, sqs=sqs))*dic%r_tot_sws
00648     dic%k3p = exp(lnk3p_sws(sal=box%salinity, tk=box%tk, tc=box%temperature, &
00649     prs=prs, prt=prt, sqs=sqs))*dic%r_tot_sws
00650     dic%kspa = 10**(lgkspa(sal=box%salinity, tk=box%tk, tc=box%temperature, prs=prs))
00651     dic%kspc = 10**(lgkspc(sal=box%salinity, tk=box%tk, tc=box%temperature, prs=prs))
00652 END SUBROUTINE dicalp
00653
00670 SUBROUTINE co2 (dic, box, niter)
00671 IMPLICIT NONE
00672 CLASS(dicsys), INTENT(INOUT) :: dic
00673 TYPE(layer), INTENT(INOUT) :: box
00674 INTEGER, INTENT(IN) :: niter
00675 CALL dicalp (dic, box) ! equilibrium constants etc.
00676 CALL dic%Hsolve (box=box, niter=niter)
00677 dic%acd = box%hc*(box%hc + dic%k1) + dic%k12
00678 box%CO2 = box%hc*box%hc*dic%ct/dic%acd
00679 box%HCO3 = box%hc*dic%ct*dic%k1/dic%acd
00680 box%CO3 = dic%ct*dic%k12/dic%acd
00681 box%omega = dic%Ca*box%CO3/dic%kspa ! Millero (1995), Eq. (81)
00682 box%omega_calcite = dic%Ca*box%CO3/dic%kspc ! Millero (1995), Eq. (80)
00683 END SUBROUTINE co2
00684
00688 SUBROUTINE hini_f06 (dic, box)
00689 IMPLICIT NONE
00690 CLASS(dicsys), INTENT(INOUT) :: dic
00691 TYPE(layer), INTENT(INOUT) :: box(0:)
00692 INTEGER :: nb
00693 box%hc = 10._dp**(-dic%pH0)
00694 DO nb=1, (SIZE(box) - 2)
00695     CALL dic%co2 (box=box(nb), niter=6)
00696 END DO
00697 END SUBROUTINE hini_f06
00698
00701 SUBROUTINE hsolve_f06 (dic, box, niter)
00702 IMPLICIT NONE
00703 CLASS(dicsys), INTENT(INOUT) :: dic
00704 TYPE(layer), INTENT(INOUT) :: box
00705 INTEGER, INTENT(IN) :: niter
00706 REAL(dp) :: k123
00707 INTEGER :: i
00708 dic%k12p = dic%k1p*dic%k2p
00709 k123 = dic%k12p*dic%k3p
00710 DO i=1,niter
00711     dic%ac = min(dic%at - dic%bt*dic%kb/(box%hc + dic%kb) - dic%kw/box%hc&
00712     - dic%pt*(box%hc*(dic%k12p - box%hc*box%hc) + 2._dp*k123)&
00713     /(box%hc*(dic%k12p + box%hc*(dic%k1p + box%hc)) + k123) &
00714     - dic%ksi*dic%sit/(box%hc + dic%ksi)&
00715     + box%hc*(dic%kSO4/(dic%kSO4 + dic%SO4) + dic%SO4/(box%hc + dic%kSO4 + dic%SO4)&
00716     + dic%ft/(box%hc + dic%kf)), 1.9_dp*dic%ct)
00717     dic%gmk = (dic%ct/dic%ac - 1._dp)*dic%k1
00718     box%hc = 0.5d0*dic%gmk + sqrt(dic%gmk*(0.25d0*dic%gmk + dic%k2) + dic%k1*dic%k2*dic%ct/dic%ac)

```

```

00719     END DO
00720 END SUBROUTINE hsolve_f06
00721
00723 SUBROUTINE dic_potalk (dic, box)
00724   IMPLICIT NONE
00725   CLASS(dicsys), INTENT(IN) :: dic
00726   TYPE(layer), INTENT(INOUT) :: box
00727   box%alkalinity = (dic%potalk*box%salinity - box%DIN)
00728   entry dic_alk(dic, box)
00729 END SUBROUTINE dic_potalk
00730
00734 REAL(dp) ELEMENTAL function eos80_rho (tmp, sal) result (rho)
00735   IMPLICIT NONE
00736   REAL(dp), INTENT(IN) :: tmp, sal
00737   rho = 999.842594e3_dp &
00738     + tmp*(67.93952_dp + tmp*(-9.09529_dp + tmp*(0.1001685_dp + tmp*(-1.120083e-3_dp +
tmp*6.536332e-6_dp)))) &
00739     + sal*(824.493_dp + tmp*(-4.0899_dp + tmp*(7.6438e-2_dp + tmp*(-8.2467e-4_dp +
tmp*5.3875e-6_dp))) &
00740     + sqrt(sal)*(-5.72466_dp + tmp*(0.10227_dp - tmp*1.6546e-3_dp)) + sal*4.8314e-1_dp)
00741 END FUNCTION eos80_rho
00742
00748 SUBROUTINE hini_m13 (dic, box)
00749   IMPLICIT NONE
00750   CLASS(dicsys), INTENT(INOUT) :: dic
00751   TYPE(layer), INTENT(INOUT) :: box(0:)
00752   INTEGER :: nb
00753   DO nb=1,(SIZE(box) - 2)
00754     CALL hset_m13 (dic, box(nb))
00755   END DO
00756 CONTAINS
00757   SUBROUTINE hset_m13 (dic, box)
00758     CLASS(dicsys), INTENT(INOUT) :: dic
00759     TYPE(layer), INTENT(INOUT) :: box
00760     REAL(dp) :: ca, ba, a0, a1, a2, dis, sdis, hmin
00761     CALL dic%alkalinity (box=box) ! set alkalinity
00762     CALL dic%alc (dic, box) ! equilibrium constants etc.
00763     IF (box%alkalinity.LE.0._dp) THEN
00764       box%hc = 1e-3_dp
00765     ELSE IF (box%alkalinity.GE.(2._dp*dic%ct + dic%bt)) THEN
00766       box%hc = 1e-10_dp
00767     ELSE
00768       ca = dic%ct/box%alkalinity
00769       ba = dic%bt/box%alkalinity
00770       ! coefficients of the cubic polynomial
00771       a0 = dic%k12*dic%kb*(1._dp - ba - 2._dp*ca)
00772       a1 = dic%k1*dic%kb*(1._dp - ba - ca) + dic%k12*(1._dp - 2._dp*ca)
00773       a2 = dic%kb*(1._dp - ba) + dic%k1*(1._dp - ca)
00774       dis = a2*a2 - 3._dp*a1 ! discriminant of the quadratic equation
00775       IF (dis > 0._dp) THEN ! if the discriminant is positive
00776         sdis = sqrt(dis)
00777         IF (a2 < 0._dp) THEN
00778           hmin = (-a2 + sdis)/3._dp
00779         ELSE
00780           hmin = -a1/(a2 + sdis)
00781         END IF
00782         box%hc = hmin + sqrt(-(a0 + hmin*(a1 + hmin*(a2 + hmin)))/sdis)
00783       ELSE
00784         box%hc = 1e-7_dp
00785       END IF
00786     END IF
00787     CALL dic%co2 (box=box, niter=4)
00788   END SUBROUTINE hset_m13
00789 END SUBROUTINE hini_m13
00790
00798 SUBROUTINE hsolve_m13 (dic, box, niter)
00799   IMPLICIT NONE
00800   CLASS(dicsys), INTENT(INOUT) :: dic
00801   TYPE(layer), INTENT(INOUT) :: box
00802   INTEGER, INTENT(IN) :: niter
00803   REAL(dp) :: delta, Hmin, Hmax, H0, H1, H2, dA0, dA1, dA2, absmin_dA,&
00804     alkinf,& !< infimum (lower bound) of non-water contributions to alkalinity
00805     alksup
00806   INTEGER :: n
00807   alkinf = -dic%pt - dic%SO4 - dic%ft ! infimum
00808   alksup = 2._dp*(dic%ct + dic%pt) + dic%bt + dic%sit ! supremum
00809   ! lower limit for [H+]
00810   delta = sqrt((dic%at - alkinf)**2 + 4._dp*dic%kw/dic%r_tot_free)
00811   IF (dic%at.GE.alkinf) THEN
00812     hmin = 2._dp*dic%kw/(dic%at - alkinf + delta)
00813   ELSE
00814     hmin = dic%r_tot_free*(alkinf - dic%at + delta)/2._dp
00815   END IF
00816   ! upper limit for [H+]
00817   delta = sqrt((dic%at - alksup)**2 + 4._dp*dic%kw/dic%r_tot_free)
00818   IF (dic%at.LE.alksup) THEN
00819     hmax = dic%r_tot_free*(alksup - dic%at + delta)/2._dp

```

```

00820     ELSE
00821         hmax = 2._dp*dic%kw/(dic%at - alksup + delta)
00822     END IF
00823     h0 = max(min(box%hc, hmax), hmin)
00824     h2 = h0
00825     da2 = deltaalk(dic, hplus=h2)
00826     absmin_da = abs(da2)
00827     IF (da2.LT.-1e-9_dp) THEN
00828         hmax = h2
00829         h1 = max(hmax/1.25_dp, sqrt(hmin*hmax))
00830     ELSE IF (da2.GT.1e-9_dp) THEN
00831         hmin = h2
00832         h1 = min(hmin*1.25_dp, sqrt(hmin*hmax))
00833     ELSE
00834         box%hc = h2
00835         RETURN
00836     END IF
00837     da1 = deltaalk(dic, hplus=h1)
00838     IF (da1.GT.1e-9_dp) THEN
00839         hmin = h1
00840     ELSE IF (da1.LT.-1e-9_dp) THEN
00841         hmax = h1
00842     ELSE
00843         box%hc = h1
00844         RETURN
00845     END IF
00846     IF (abs(da1).GT.absmin_da) THEN
00847         h0 = h1
00848         da0 = da1
00849         h1 = h2
00850         da1 = da2
00851         h2 = h0
00852         da2 = da0
00853     ELSE
00854         absmin_da = abs(da1)
00855     END IF
00856     h0 = h1 - da1*(h2 - h1)/(da2 - da1)
00857     IF (h0.GT.hmax) h0 = sqrt(h1*hmax)
00858     IF (h0.LT.hmin) h0 = sqrt(h1*hmin)
00859     DO n=1,niter
00860         da0 = deltaalk(dic, hplus=h0)
00861         IF (da0.GT.0._dp) hmin = h0
00862         IF (da0.LT.0._dp) hmax = h0
00863         h2 = h1
00864         da2 = da1
00865         h1 = h0
00866         da1 = da0
00867         IF (abs(da0).GE.0.5_dp*absmin_da) THEN
00868             h0 = sqrt(hmax*hmin)
00869         ELSE
00870             h0 = h1 - da1*(h2 - h1)/(da2 - da1)
00871             IF (h0.LT.hmin) h0 = sqrt(h1*hmin)
00872             IF (h0.GT.hmax) h0 = sqrt(h1*hmax)
00873         END IF
00874         absmin_da = min(abs(da0), absmin_da)
00875     END DO
00876     box%hc = h0
00877 CONTAINS
00878 FUNCTION deltaalk (dic, hplus) RESULT (dAlk)
00879 IMPLICIT NONE
00880 CLASS(dicsys), INTENT(IN) :: dic
00881 REAL(dp), INTENT(IN) :: hplus
00882 REAL(dp) :: dalk, alkdic, alkbtor, alkp04, alksil, alkso4, alkf, alkh2o
00883 alkdic = dic%ct*(2._dp*dic%k12 + hplus*dic%k1)/(dic%k12 + hplus*(dic%k1 + hplus))
00884 alkbtor = dic%bt*dic%kb/(dic%kb + hplus)
00885 alkp04 = dic%pt*(3._dp*dic%k3p + hplus*(2._dp*dic%k2p + hplus*dic%k1p))&
00886         / (dic%k3p + hplus*(dic%k2p + hplus*(dic%k1p + hplus)))
00887 alksil = dic%sit*dic%ksi/(dic%ksi + hplus)
00888 alkso4 = dic%SO4*(dic%kSO4/(dic%kSO4 + hplus) - 1._dp)
00889 alkf = dic%ft*(dic%kf/(dic%kf + hplus) - 1._dp)
00890 alkh2o = dic%kw/hplus - hplus/dic%r_tot_free
00891 dalk = alkdic + alkbtor + alkp04 + alksil + alkso4 + alkf + alkh2o - dic%at
00892 END FUNCTION deltaalk
00893 END SUBROUTINE hsolve_m13
00894 END MODULE dic

```

7.15 /Users/mpahlow/oppla/src/dvode.f90 File Reference

Data Types

- type `dvode::ode`

Modules

- module `dcode`
VODE_F90 interface for oppla.

Functions/Subroutines

- subroutine `dcode::read_ode` (parode, lun, nconstr)
Read DVODE parameters.
- subroutine `dcode::set_atol` (parode, stv)
Set absolute tolerances for each state individually.

7.16 dcode.f90

[Go to the documentation of this file.](#)

```

00001
00002 MODULE dcode
00003   USE dcode_f90_m
00004   USE netcdf, ONLY : nf90_max_name
00005   USE stdunt
00006   IMPLICIT NONE
00007   PRIVATE
00008   PUBLIC :: vcode_opts, set_opts, get_stats, vcode_f90, xsetf
00009   TYPE, PUBLIC :: ode
00010     REAL(dp), DIMENSION(:), ALLOCATABLE :: atol,& !< absolute tolerances
00011     REAL(dp), DIMENSION(:), ALLOCATABLE :: rtol,& !< relative tolerances
00012     REAL(dp), DIMENSION(:), ALLOCATABLE :: mxatol,& !< maxima of atol
00013     REAL(dp), DIMENSION(:), ALLOCATABLE :: mxrtol,& !< maxima of rtol
00014     REAL(dp), DIMENSION(:), ALLOCATABLE :: clower,& !< lower constraints
00015     REAL(dp), DIMENSION(:), ALLOCATABLE :: cupper
00016     REAL(dp) :: h0=0d0, hmin=0d0, hmax=0d0
00017     INTEGER :: itask=4, mf, mxstep=500, mxhnil=0, maxord=12, meth=1, miter=3,&
00018     jsv=1, neg=0, nconstr=0, moss=0
00019     INTEGER, DIMENSION(:), ALLOCATABLE :: constrained
00020     CHARACTER(LEN=nf90_max_name), DIMENSION(:), ALLOCATABLE :: names
00021     LOGICAL :: quiet=.false., constr=.true., sparse_jacobian
00022   CONTAINS
00023     PROCEDURE :: read => read_ode
00024     PROCEDURE :: set_atol => set_atol
00025   END TYPE ode
00026
00027 CONTAINS
00028
00030 SUBROUTINE read_ode (parode, lun, nconstr)
00031   IMPLICIT NONE
00032   CLASS(ode), INTENT(INOUT) :: parode
00033   INTEGER, INTENT(IN) :: lun
00034   INTEGER, INTENT(IN) :: nconstr
00035   CHARACTER(LEN=512) :: msg="", filename
00036   REAL(dp) :: atol, rtol, mxatol, mxrtol, h0, hmin, hmax
00037   INTEGER :: itask, mxstep, mxhnil, maxord, meth, miter, jsv, n
00038   LOGICAL :: quiet, constr
00039   namelist /vode/ atol, rtol, h0, hmin, hmax, quiet, constr, itask,&
00040   mxrtol, mxatol, mxstep, mxhnil, maxord, meth, miter, jsv
00041   parode%nconstr = nconstr
00042   atol = 1.e-9_dp
00043   mxatol = 0._dp
00044   rtol = 1.e-6_dp
00045   mxrtol = 1.e-5_dp
00046   h0 = parode%h0
00047   hmin = parode%hmin
00048   hmax = parode%hmax
00049   quiet = parode%quiet
00050   constr = parode%constr
00051   itask = parode%itask
00052   jsv = parode%jsv
00053   mxstep = parode%mxstep
00054   mxhnil = parode%mxhnil
00055   maxord = parode%maxord
00056   meth = parode%meth
00057   miter = parode%miter
00058   rewind(unit=lun)
00059   READ (unit=lun, nml=vode, END=10, IOMSG=msg, ERR=900)

```

```

00060      IF (meth.EQ.1) maxord = min(maxord, 12)
00061      IF (meth.EQ.2) maxord = min(maxord, 5)
00062      parode%h0 = h0
00063      parode%hmin = hmin
00064      parode%hmax = hmax
00065      parode%quiet = quiet
00066      parode%constr = constr
00067      parode%itask = itask
00068      parode%jsv = jsv
00069      parode%mxstep = mxstep
00070      parode%mxhnil = mxhnil
00071      parode%maxord = maxord
00072      parode%meth = meth
00073      parode%miter = miter
00074 10  CONTINUE
00075      IF (len_trim(msg).GT.0) THEN
00076          INQUIRE (unit=lun, name=filename)
00077          WRITE (stderr,'("read_ode: namelist vode not in ",A,".")') trim(filename)
00078      END IF
00079      parode%sparse_jacobian = miter.EQ.7
00080      IF (parode%sparse_jacobian) parode%moos = 2
00081      parode%mf = parode%moos*100 + jsv*(10*meth + miter)
00082      IF (ALLOCATED(parode%constrained)) &
00083          DEALLOCATE (parode%constrained, parode%clower, parode%cupper)
00084      ALLOCATE (parode%atol(parode%neq), parode%mxatol(parode%neq), &
00085          parode%rtol(parode%neq), parode%mxrtol(parode%neq), &
00086          parode%constrained(parode%nconstr), &
00087          parode%clower(parode%nconstr), parode%cupper(parode%nconstr))
00088      parode%constrained = [(n, n=1, parode%nconstr)] ! all plankton states are constrained
00089      parode%atol = atol
00090      IF (mxatol.LE.0._dp) THEN
00091          parode%mxatol = atol * 10._dp
00092      ELSE
00093          parode%mxatol = mxatol
00094      END IF
00095      parode%rtol = rtol
00096      IF (mxrtol.LE.0._dp) THEN
00097          parode%mxrtol = rtol * 10._dp
00098      ELSE
00099          parode%mxrtol = mxrtol
00100      END IF
00101      parode%clower = 0d0
00102      parode%cupper = 1d6
00103      RETURN
00104 900 CONTINUE
00105      WRITE (stderr,'("read_ode:",A)') trim(msg)
00106      stop 1
00107  END SUBROUTINE read_ode
00108
00113  SUBROUTINE set_atol (parode, stv)
00114      IMPLICIT NONE
00115      CLASS(ode), INTENT(INOUT) :: parode
00116      REAL(dp), INTENT(IN) :: stv(*)
00117      IF (all(parode%atol.LT.0.0_dp)) THEN
00118          parode%atol(1:parode%nconstr) = -parode%atol(1:parode%nconstr)*stv(1:parode%nconstr)
00119          parode%atol(parode%nconstr+1:parode%neq) = abs(parode%atol(parode%nconstr+1:parode%neq))
00120      END IF
00121      IF (all(parode%mxatol.LT.0.0_dp)) THEN
00122          parode%mxatol(1:parode%nconstr) = -parode%mxatol(1:parode%nconstr)*stv(1:parode%nconstr)
00123          parode%mxatol(parode%nconstr+1:parode%neq) = abs(parode%mxatol(parode%nconstr+1:parode%neq))
00124      END IF
00125  END SUBROUTINE set_atol
00126
00127  END MODULE dvode

```

7.17 /Users/mpahlow/oppla/src/et.f90 File Reference

Data Types

- interface [et::aggregation](#)
Calculate aggregation fluxes.
- interface [et::attenuate](#)
Light attenuation.
- interface [et::boxalk](#)
- interface [et::boxbc](#)
- interface [et::decl](#)

- daylength function*
- interface [et::fluxes](#)
 - Calculate fluxes in the source matrix soma.*
- interface [et::fpar](#)
 - Fractions of surface PAR arriving at the box bottoms.*
- type [et::fungroup](#)
- interface [et::init](#)
 - Read namelists.*
- type [et::layer](#)
 - Conditions and fluxes local to each box (layer) [More...](#)*
- interface [et::light](#)
 - Calculate or read surface irradiance.*
- type [et::local](#)
- interface [et::preset](#)
 - Define ratios, set indices, pointers.*
- type [et::state](#)
- interface [et::sunday](#)
- type [et::timing](#)

Modules

- module [et](#)
 - Ecological types.*

Functions/Subroutines

- real([dp](#)) function [et::ft_eppley](#) (grp, box)
 - Temperature function after Eppley (1972)*
- real([dp](#)) function [et::ft_houlton](#) (grp, box)
 - Temperature function after Houlton et al. (2008)*
- real([dp](#)) function [et::ft_me](#) (grp, box)
 - Temperature function for Mytilus edulis.*
- real([dp](#)) function [et::ft_oppla](#) (grp, box)
 - Temperature function based on Moisan et al. (2002)*
- subroutine [et::ft_select](#) (grp, ft, caller)
 - Select and associate temperature-dependence function.*

Variables

- character(len= *), dimension(*), parameter [et::constituent_units](#) =(/'mmol m-3 ', 'mmol m-3 ', 'mmol m-3 ', 'mmol m-3 ', 'mgChl m-3'/)
- character(len= *), dimension(*), parameter [et::constituents](#) =(/'C ', 'N ', 'P ', 'PIC', 'Chl'/)
- real([dp](#)), parameter [et::pi](#) =3.14159265358979323_dp
 - Common components for all functional groups.*
- real([dp](#)), parameter [et::rad](#) =pi/180._dp

7.17.1 Data Type Documentation

7.17.1.1 type et::layer

Conditions and fluxes local to each box (layer)

- stv is a pointer to the state variables in the current box, i.e., [stv\(nb,:\)](#) in module plankton and stvdat(nb,:) in subroutine invar (module onf)
- soma is a pointer to the source matrix of fluxes among state variables within the current box (layer); the first index denotes the source, the second index the target of the flux; diagonal elements store rates of change not affecting other state variables; the first row (first index = 0) is for aggregation

Definition at line 146 of file [et.f90](#).

Class Members

real(dp), pointer	alkalinity	
real(dp), dimension(:), allocatable	bbc	bottom boundary conditions
real(dp)	botperm =1._dp	bottom permeability (for sinking etc.)
real(dp), pointer	bpar	irradiance (PAR) at bottom of box
real(dp)	co2 =0._dp	CO2 concentration in mol kg ⁻¹ .
real(dp)	co3	CO3 concentration in mol kg ⁻¹ .
real(dp)	delvel	velocity difference between top and bottom of box
real(dp)	depth	mid-depth of current box in m
real(dp), pointer	dic	
real(dp), pointer	din	
real(dp), pointer	dip	
real(dp)	dpar	depth average of daytime irradiance
real(dp)	dz	distance between current and above boxes mid-depths
integer	fish =0	factor for fish mortality (0 or 1)
real(dp), dimension(:), pointer	flux	vector of vertical fluxes toward the layer beneath
real(dp)	hc	proton concentration in mol kg ⁻¹
real(dp)	hco3	HCO3 concentration in mol kg ⁻¹ .
real(dp)	height =0D0	height of current box in m
real(dp)	ipar	instantaneous depth-averaged irradiance
real(dp)	lch	(light-attenuation coefficient)*(box height)
integer	nb	number of current box (layer); nb = 1 is the surface box
real(dp), pointer	o2	
real(dp)	omega	aragonite saturation
real(dp)	omega_calcite	calcite saturation
real(dp), dimension(:), pointer	ratios	ratios in this layer
real(dp)	rdl	ratio of lower box height to total height of two boxes
real(dp), pointer	rdoc	
real(dp), pointer	rdon	
real(dp)	rho	density in g m ⁻³
real(dp), pointer	salinity	
real(dp), pointer	silc	total Si concentration in mol kg ⁻¹
real(dp), dimension(:, :), pointer	soma	fluxes among states within the current box

Class Members

real(dp), dimension(:), pointer	stv	state variables in the current box
real(dp), pointer	temperature	
real(dp)	tk	temperature in K
real(dp), pointer	tpar	irradiance (PAR) at top of box
real(dp), pointer	vdc	vertical diffusivity in $\text{m}^2 \text{s}^{-1}$
real(dp), pointer	velocity	vertical velocity in m s^{-1} (positive downward)

7.17.1.2 type et::local

Definition at line 63 of file [et.f90](#).

Class Members

real(dp), dimension(:), allocatable	agg	
procedure(agggregation), pointer	aggregate	aggregation fluxes
procedure(boxalk), pointer	alkalinity	alkalinity
real(dp), dimension(:, :), allocatable	all_ratios	all ratios in all boxes
procedure(attenuate), pointer	attenuation	light attenuation
real(dp), dimension(:), allocatable	bbc	bottom-boundary conditions for layers
procedure(boxbc), pointer	boxbc	box-boundary concentrations/conditions
logical	calcpar = .TRUE.	should PAR be calculated from clear-sky radiation in sunday_brock81?
real(dp), pointer	daylen	day length fraction
character(len=256)	daylenfun = "	
real(dp), pointer	daypar	average day-time surface PAR
real(dp), dimension(:), allocatable	depth	
real(dp), dimension(:, :), pointer	depth_edges	depths of layer edges
character(len=256)	dicy = "	diurnal light cycle
real(dp), dimension(:), allocatable	dydz	
real(dp), dimension(:), allocatable	falk	alkalinity factors for DIN, PIC, etc.
real(dp), dimension(:, :), allocatable	fluxes	flux vectors
logical, dimension(:, :), allocatable	fmsk	fluxes-matrix mask
real(dp), dimension(:), allocatable	fpar	fraction of sPAR arriving at the top of each layer
real(dp), dimension(:), allocatable	fsfalk	alkalinity factors for surface fluxes
real(dp), dimension(:, :), allocatable	heights	layer heights
integer, dimension(:), allocatable	iagg	
integer	ialk = 0	
real(dp), pointer	ice	ice cover fraction
integer, dimension(:), allocatable	ichl	indices of Chl tracers
integer	idagg = 0	index of dretritus group representing aggregates
integer	idic = 0	
integer	idin = 0	
integer	idip = 0	
integer, dimension(:), allocatable	ifalk	indices of tracers influencing alkalinity
integer	io2 = 0	
integer, dimension(:), allocatable	iphy	

Class Members

integer, dimension(:), allocatable	ipic	indices of PIC tracers
integer, dimension(:), allocatable	ipoc	indices of POC tracers
integer, dimension(:), allocatable	iq0r	ratio-indices of subsistence quotas
integer, dimension(:), allocatable	irc	indices of C tracers (reference tracers for ratios)
integer	irdoc =0	
integer	irdon =0	
integer, dimension(:), allocatable	irt	tracer-indices of quota tracers
integer, dimension(:), allocatable	isfalk	indices of surface fluxes influencing alkalinity
integer	isi =0	
real(dp), dimension(:,,:), allocatable	kij	coagulation kernel matrix
real(dp)	lachi =16D0	
real(dp), pointer	lacw	light-attenuation coefficient of water in m^{-1}
real(dp)	lapon =16D0	light attenuation coefficient of chl, pon
integer	nagg	No. of types involved in aggregation.
integer	nbac	
integer	nbox	No. of boxes (layers)
integer	ncon	No. of different material constituents.
integer	ndet	
integer	ndm	No. of dissolved-matter variables except labile DOM.
integer	ndom	No. of DOM groups.
integer	nft	No. of functional types.
integer	nphy	
integer	nq0	No. of subsistence quotas.
integer	nrs	No. of quotas.
integer	nzoo	No. of bacteria, detritus, phytoplankton, zooplankton groups.
real(dp), dimension(:), allocatable	oc	DOC or POC for all groups.
real(dp), dimension(:), allocatable	par	surface PAR for each layer
procedure(fpar), pointer	parfrac	fractions of surface PAR arriving at the box bottoms
procedure(light), pointer	parfun	calculate or read surface irradiance
real(dp), pointer	pressure	atmospheric surface pressure in Pa
real(dp), dimension(:), allocatable	q0	subsistence quotas
logical, dimension(:,,:), allocatable	qmsk	quota mask: .TRUE. if the quota is variable (a dynamic ratio)
real(dp), dimension(:,,:), allocatable	quotas	biomass (C)-normalised quotas for all groups, dimensions are (nq,nft): <code>quotas(1,:) = 1</code> , <code>quotas(2,:) = QN</code> , <code>quotas(3,:) = QP</code> , etc.
real(dp), dimension(:), allocatable	ratios	ratios of all groups
real(dp)	rq =1.2D0	respiratory quotient in $\text{mol O}_2 (\text{mol C})^{-1}$
real(dp)	salinity	constant salinity if no profile data available
real(dp), pointer	sfdic	surface flux of DIC in $\text{mmol m}^{-2} \text{s}^{-1}$
real(dp), pointer	sfo2	surface flux of O_2 in $\text{mmol m}^{-2} \text{s}^{-1}$
logical, dimension(:,,:), allocatable	smsk	source-matrix mask
real(dp), dimension(:,,:), allocatable	soma	source matrices
real(dp), pointer	spar	instantaneous surface PAR

Class Members

real(dp), dimension(:), allocatable	sticky	
real(dp)	temperature	constant temperature if no profile data available
real(dp)	vdc	constant vertical mixing if no profile data available
real(dp)	velocity	constant vertical velocity if no profile data available
real(dp), dimension(:), allocatable	vvs	relative vertical velocities (sinking, vertical migration, etc.)
real(dp), dimension(:), allocatable	vvstv	absolute vertical velocities, including up- or downwelling
real(dp), pointer	wind	wind speed in m s^{-1}
real(dp), pointer	xco2	mole fraction of CO_2 in air
logical	xsfolk	do any surface fluxes affect alkalinity?

7.17.1.3 type et::state

Definition at line 60 of file [et.f90](#).

Class Members

class(fungroup), pointer	var	pointer to functional group of type bacteria, phycmo, zoocfo, etc.
--------------------------	-----	--

7.17.1.4 type et::timing

Definition at line 179 of file [et.f90](#).

Class Members

real(dp)	cdl	$\text{COS}(d1) * \text{COS}(\text{latrad})$
integer(int64)	current	current model time in s during integration in subroutine plankton:plankton_ode
real(dp)	d1	declination
real(dp)	daylen = 0.5_dp	day length fraction
procedure(decl), pointer	daylength	day length function
procedure(decl), pointer	declination	declination function
real(dp), dimension(:), allocatable	dil_din	
real(dp), dimension(:), allocatable	dil_dip	
real(dp), dimension(:), allocatable	dil_fact	
real(dp), dimension(:), allocatable	dil_time	
logical	dilute = .FALSE.	
integer	doy	day of year
real(dp)	dto = 1D0	output time step (not related to time step for integration)
integer	dtutc	difference between local time and UTC in s
real(dp)	end	time of end of simulation
real(dp)	hour	current hour of the day
real(dp)	latdeg	

Class Members

real(dp), pointer	latrad	
real(dp)	londeg	
integer	ndil =0	number of dilution events
real(dp)	noon =12._dp*cfhs	noon-time of day in s
real(dp)	now	current model time in s in the main program
real(dp)	previous	model time at previous output
real(dp)	ptl =0.8333_dp	twilight parameter in degrees
real(dp)	sample =0D0	
real(dp)	start =0._dp	time of start of simulation
integer(int64), dimension(4)	sun	current-day sunrise, noon, sunset, midnight in s
procedure(sunday), pointer	sunday	
integer(kind=int64), pointer	sunrise	
integer(kind=int64), pointer	sunset	
integer(int64)	tbase	time 0 in s with respect to date units in forcing file
integer	timeout =0	
real(dp)	tin =0D0	time of initial conditions in initial-conditions file
real(dp)	toy	time of year in decimal days
real(dp)	w1	sunset hour angle

7.18 et.f90

[Go to the documentation of this file.](#)

```

00001
00014 MODULE et
00015   USE stdunt
00016   USE fudu
00017   USE iso_fortran_env, ONLY: int64
00018   IMPLICIT NONE
00028   REAL(dp), PARAMETER :: pi=3.14159265358979323_dp, rad=pi/180._dp
00029   CHARACTER(LEN=*) , PARAMETER :: constituents(*)=(/'C ', 'N ', 'P ', 'PIC', 'Chl'/), &
00030     constituent_units(*)=(/'mmol m-3 ', 'mmol m-3 ', 'mmol m-3 ', 'mmol m-3 ', 'mgChl m-3'/)
00031   TYPE, ABSTRACT :: fungroup
00032     CHARACTER(LEN=512) :: name
00033     CHARACTER(LEN=100), ALLOCATABLE :: names(:), & !< names of state variables
00034     units(:), & !< units of state variables
00035     constituents(:)
00036     INTEGER :: ifg, & !< index of functional group
00037     ncon, & !< number of constituents
00038     nsv=1, & !< number of state variables
00039     nq0=0
00040     INTEGER, ALLOCATABLE :: isv(:), & !< vector of state-variable indices
00041     icon(:), & !< local constituent indices
00042     ir(:), & !< global ratio indices
00043     iq0(:)
00044     LOGICAL :: motile=.false.
00045     REAL(dp), ALLOCATABLE :: q0(:)
00046     REAL(dp), POINTER :: oc, & !< organic C
00047     qn, & !< N:C ratio (cell quota)
00048     qp, & !< P:C ratio (cell quota)
00049     stick, & !< stickiness (\unit{m^3.(mmol.C)^{-1}})
00050     vv(:), & !< vertical velocities
00051     ratios(:)
00052     REAL(dp) :: q10=1.89_dp, tref=27._dp, topt=15._dp, tspr=12._dp, sticky=0._dp
00053     PROCEDURE(ft_eppley), POINTER :: ft => ft_eppley ! temperature function
00054   CONTAINS
00055     PROCEDURE :: ft_select => ft_select
00056     PROCEDURE(fluxes), DEFERRED :: flux
00057     PROCEDURE(init), DEFERRED :: read
00058     PROCEDURE(preset), DEFERRED :: set
00059   END TYPE fungroup
00060   TYPE :: state
00061     CLASS(fungroup), POINTER :: var

```

```

00062 END TYPE state
00063 TYPE :: local
00064   INTEGER :: idic=0, io2=0, idin=0, idip=0, ialk=0, isi=0, irdoc=0, irdon=0,&
00065     idagg=0,&
00066     nft,&
00067     nbox,&
00068     ncon,&
00069     nq0,&
00070     nrs,&
00071     ndm,&
00072     ndom,&
00073     nagg,&
00074     nbac, ndet, nphy, nzoo
00075   INTEGER, DIMENSION(:), ALLOCATABLE :: &
00076     iq0r,&           !< ratio-indices of subsistence quotas
00077     irc,&             !< indices of C tracers (reference tracers for ratios)
00078     irt,&             !< tracer-indices of quota tracers
00079     ichl,&            !< indices of Chl tracers
00080     ipic,&            !< indices of PIC tracers
00081     ipoc,&            !< indices of POC tracers
00082     ifalk,&           !< indices of tracers influencing alkalinity
00083     isfalk,&          !< indices of surface fluxes influencing alkalinity
00084     iphy, iagg
00085   CHARACTER(LEN=256) :: dicy="," !< diurnal light cycle
00086   daylenfun="
00087   LOGICAL :: calcpa=.true.,&
00088     xsfalk
00090   LOGICAL, ALLOCATABLE :: qmsk(:,:),&
00091     fmsk(:,:),&           !< fluxes-matrix mask
00092     smask(:,:),&
00093   REAL(dp), DIMENSION(:,:), ALLOCATABLE :: soma
00096   REAL(dp), DIMENSION(:,:), ALLOCATABLE :: quotas,&
00097     fluxes,&              !< flux vectors
00098     heights,&             !< layer heights
00099     all_ratios,&          !< all ratios in all boxes
00100     kij
00101   REAL(dp), DIMENSION(:), ALLOCATABLE :: q0,& !< subsistence quotas
00102     ratios,&              !< ratios of all groups
00103     par,&                 !< surface PAR for each layer
00104     fpar,&               !< fraction of sPAR arriving at the top of each layer
00105     falk,&               !< alkalinity factors for DIN, PIC, etc.
00106     fsfalk,&             !< alkalinity factors for surface fluxes
00107     oc,&                 !< DOC or POC for all groups
00108     sticky, agg,&
00109     depth,&
00110     vvs,&                !< relative vertical velocities (sinking, vertical migration, etc.)
00111     vvstv,&              !< absolute vertical velocities, including up- or downwelling
00112     dydz,&
00113     bbc
00114   REAL(dp), POINTER :: daypar,& !< average day-time surface PAR
00115     spar,&               !< instantaneous surface PAR
00116     lacw,&               !< light-attenuation coefficient of water in \unit{m^{-1}}
00117     pressure,&           !< atmospheric surface pressure in Pa
00118     xco2,&               !< mole fraction of \ce{CO2} in air
00119     sfdic,&              !< surface flux of DIC in \unit{mmol.m^{-2}.s^{-1}}
00120     sfo2,&               !< surface flux of \ce{O2} in \unit{mmol.m^{-2}.s^{-1}}
00121     wind,&               !< wind speed in \unit{m.s^{-1}}
00122     daylen,&             !< day length fraction
00123     ice,&                !< ice cover fraction
00124     depth_edges(:,:)
00125   REAL(dp) :: lachl=16d0, lapon=16d0,&
00126     rq=1.2d0,&
00127     temperature,&
00128     salinity,&
00129     vdc,&
00130     velocity
00131   PROCEDURE(light), POINTER :: parfun
00132   PROCEDURE(attenuate), POINTER :: attenuation
00133   PROCEDURE(fpar), POINTER :: parfrac
00134   PROCEDURE(boxbc), POINTER :: boxbc
00135   PROCEDURE(boxalk), POINTER :: alkalinity
00136   PROCEDURE(aggregation), POINTER :: aggregate
00137 END TYPE local
00146 TYPE :: layer
00147   INTEGER :: nb,&        !< number of current box (layer); nb = 1 is the surface box
00148     fish=0
00149   REAL(dp) :: dz,&      !< distance between current and above boxes mid-depths
00150     depth,&             !< mid-depth of current box in m
00151     height=0d0,&
00152     lch,&
00153     ipar,&
00154     dpar,&
00155     delvel,&
00156     rdl,&
00157     hc,&
00158     co2=0._dp,&
00159     hco3,&

```

```

00160         co3,&
00161         rho,&
00162         omega_calcite,&
00163         omega,&
00164         tk,&
00165         botperm=1._dp
00166     REAL(dp), ALLOCATABLE :: bbc(:)
00167     REAL(dp), POINTER :: tpar,&          !< irradiance (PAR) at top of box
00168         bpar,&                          !< irradiance (PAR) at bottom of box
00169         temperature, salinity, alkalinity,&
00170         stv(:),&                        !< state variables in the current box
00171         din, dip, dic, o2, rdoc, rdon,&
00172         silc,&                          !< total Si concentration in \unit{mol.kg^{-1}}
00173         vdc,&                          !< vertical diffusivity in \unit{m^2.s^{-1}}
00174         velocity,&                      !< vertical velocity in \unit{m.s^{-1}} (positive downward)
00175         ratios(:),&                    !< ratios in this layer
00176         flux(:),&                      !< vector of vertical fluxes toward the layer beneath
00177         soma(:,:)
00178 END TYPE layer
00179 TYPE, PUBLIC :: timing
00180     REAL(dp) :: londeg, latdeg,&
00181         now,&                          !< current model time in s in the main program
00182         previous,&                     !< model time at previous output
00183         start=0._dp,&
00184         end,&                          !< time of end of simulation
00185         hour,&
00186         sample=0d0,&
00187         daylen=0.5_dp,&
00188         ptl=0.8333_dp,&
00189         wl,&
00190         dl,&
00191         cdl,&
00192         dto=1d0,&
00193         tin=0d0,&
00194         toy,&
00195         noon=12._dp*cfhs
00196     REAL(dp), POINTER :: latrad
00197     REAL(dp), DIMENSION(:), ALLOCATABLE :: dil_time, dil_fact, dil_din, dil_dip
00198     INTEGER :: doy,&                  !< day of year
00199         dtutc,&                       !< difference between local time and UTC in s
00200         ndil=0,&                     !< number of dilution events
00201         timeout=0
00202     INTEGER(INT64) :: current,& !< current model time in s during integration in subroutine
plankton:plankton_ode
00203         sun(4),&                     !< current-day sunrise, noon, sunset, midnight in s
00204         tbase
00205     INTEGER(KIND=INT64), POINTER :: sunrise, sunset
00206     LOGICAL :: dilute=.false.
00207     PROCEDURE(sunday), POINTER :: sunday
00208     PROCEDURE(decl), POINTER :: declination,& !< declination function
00209         daylength
00210 END TYPE timing
00211
00212 abstract INTERFACE
00213
00214     SUBROUTINE fluxes (grp, grps, env, box, times)
00215         IMPORT dp, fungroup, state, local, layer, timing
00216         IMPLICIT NONE
00217         CLASS(fungroup), INTENT(INOUT) :: grp
00218         TYPE(state), DIMENSION(:), INTENT(IN) :: grps
00219         TYPE(local), INTENT(INOUT) :: env
00220         TYPE(layer), INTENT(INOUT) :: box(:)
00221         TYPE(timing), INTENT(IN) :: times
00222     END SUBROUTINE fluxes
00223
00224     SUBROUTINE init (grp, env, lun)
00225         IMPORT fungroup, local
00226         IMPLICIT NONE
00227         CLASS(fungroup), TARGET, INTENT(INOUT) :: grp
00228         TYPE(local), TARGET, INTENT(IN) :: env
00229         INTEGER, INTENT(IN) :: lun
00230     END SUBROUTINE init
00231
00232     SUBROUTINE preset (grp, grps, env)
00233         IMPORT dp, fungroup, state, local
00234         IMPLICIT NONE
00235         CLASS(fungroup), TARGET, INTENT(INOUT) :: grp
00236         TYPE(state), DIMENSION(:), INTENT(IN) :: grps
00237         TYPE(local), TARGET, INTENT(INOUT) :: env
00238     END SUBROUTINE preset
00239
00240     SUBROUTINE aggregation (env, grps, box)
00241         IMPORT dp, local, state, layer
00242         IMPLICIT NONE
00243         CLASS(local), INTENT(INOUT) :: env
00244         TYPE(state), INTENT(IN) :: grps(:)
00245         TYPE(layer), INTENT(INOUT) :: box
00246     END SUBROUTINE aggregation
00247
00248     SUBROUTINE light (ambient, time)
00249         IMPORT local, timing

```



```

00250      IMPLICIT NONE
00251      CLASS(local), INTENT(INOUT) :: ambient
00252      TYPE(timing), INTENT(INOUT) :: time
00253  END SUBROUTINE light
00255  SUBROUTINE attenuate (ambient, boxes, stv)
00256      IMPORT dp, local, layer
00257      IMPLICIT NONE
00258      CLASS(local), INTENT(IN) :: ambient
00259      TYPE(layer), INTENT(INOUT) :: boxes(:)
00260      REAL(dp), INTENT(IN) :: stv(:, :)
00261  END SUBROUTINE attenuate
00263  SUBROUTINE fpar (env, boxes, nbox)
00264      IMPORT local, layer
00265      CLASS(local), INTENT(INOUT) :: env
00266      TYPE(layer), INTENT(INOUT) :: boxes(0:)
00267      INTEGER, INTENT(IN) :: nbox
00268  END SUBROUTINE fpar
00269  SUBROUTINE boxbc (env, boxes)
00270      IMPORT dp, local, layer
00271      IMPLICIT NONE
00272      CLASS(local), INTENT(IN) :: env
00273      TYPE(layer), INTENT(INOUT) :: boxes(:)
00274  END SUBROUTINE boxbc
00275  SUBROUTINE boxalk (env, thisbox)
00276      IMPORT dp, local, layer
00277      IMPLICIT NONE
00278      CLASS(local), INTENT(IN) :: env
00279      TYPE(layer), INTENT(INOUT) :: thisbox
00280  END SUBROUTINE boxalk
00281  SUBROUTINE sunday (times, env)
00282      IMPORT local, timing
00283      CLASS(timing), INTENT(INOUT) :: times
00284      TYPE(local), INTENT(INOUT) :: env
00285  END SUBROUTINE sunday
00287  SUBROUTINE decl (times)
00288      IMPORT dp, timing
00289      IMPLICIT NONE
00290      CLASS(timing), INTENT(INOUT) :: times
00291  END SUBROUTINE decl
00292  END INTERFACE
00293  CONTAINS
00295  FUNCTION ft_eppley (grp, box) RESULT (fT)
00296      IMPLICIT NONE
00297      CLASS(funigroup), INTENT(IN) :: grp
00298      TYPE(layer), INTENT(IN) :: box
00299      REAL(dp) :: fT
00300      fT = grp%Q10**((0.1_dp*(box%temperature - grp%Tref))
00301  END FUNCTION ft_eppley
00303  FUNCTION ft_houlton (grp, box) RESULT (ftp)
00304      IMPLICIT NONE
00305      CLASS(funigroup), INTENT(IN) :: grp
00306      TYPE(layer), INTENT(IN) :: box
00307      REAL(dp) :: ftp
00308      ftp = exp(-3.62d0 + 0.27d0*box%temperature*(1d0 - box%temperature/(2d0*grp%Tref)))
00309  END FUNCTION ft_houlton
00311  FUNCTION ft_oppla (grp, box) RESULT (fT)
00312      IMPLICIT NONE
00313      CLASS(funigroup), INTENT(IN) :: grp
00314      TYPE(layer), INTENT(IN) :: box
00315      REAL(dp) :: fT
00316      fT = grp%Q10**((0.1_dp*(box%temperature - grp%Tref)) &
00317      *exp(-(abs(box%temperature - grp%Topt)/grp%Tspr)**3))
00318  END FUNCTION ft_oppla
00320  FUNCTION ft_me (grp, box) RESULT (fT)
00321      IMPLICIT NONE
00322      CLASS(funigroup), INTENT(IN) :: grp
00323      TYPE(layer), INTENT(IN) :: box
00324      REAL(dp) :: fT
00325      fT = (1d0 + (box%temperature - 15d0)*9.5d0)/260d0
00326  END FUNCTION ft_me
00328  SUBROUTINE ft_select (grp, fT, caller)
00329      IMPLICIT NONE
00330      CLASS(funigroup), INTENT(INOUT) :: grp
00331      CHARACTER(LEN=*) , INTENT(IN) :: fT, & !< name of temperature function ('oppla', 'Eppley',
'< name of temperature function ('oppla', 'Eppley',
00332      caller
00333      SELECT CASE (trim(fT))
00334      CASE ('Eppley', '')
00335          grp%fT => ft_eppley
00336      CASE ('Houlton')
00337          grp%fT => ft_houlton
00338      CASE ('oppla')
00339          grp%fT => ft_oppla
00340      CASE ('ME')
00341          grp%fT => ft_me
00342      CASE DEFAULT
00343          WRITE (stderr, ' ("ft_select, called by ",A,": invalid temperature function ",A,".")')

```

```

        trim(caller), trim(ft)
00344      stop 1
00345      END SELECT
00346  END SUBROUTINE ft_select
00347 END MODULE et

```

7.19 /Users/mpahlow/oppla/src/fudu.F90 File Reference

Data Types

- interface [fudu::cv_convert](#)
- interface [fudu::cv_free](#)
- type [fudu::quantity](#)

Type for keeping values and units together; enables easy input of units in namelists. [More...](#)

- interface [fudu::ut_decode_time](#)
- interface [fudu::ut_encode_time](#)
- interface [fudu::ut_free](#)
- interface [fudu::ut_free_system](#)
- interface [fudu::ut_get_converter](#)
- interface [fudu::ut_get_status](#)
- interface [fudu::ut_parse](#)

Parse unit string.

- interface [fudu::ut_read_xml](#)

Set-up and return units system from xml database.

Modules

- module [fudu](#)

FORTTRAN interface for the UDUNITS2 library.

Functions/Subroutines

- [real\(c_double\) function, public \[fudu::convert\]\(#\) \(qty, unit\)](#)

Convert a quantity to another unit.

- subroutine, public [fudu::exudu](#)

Unregister the units database associated with the global variable utsystem.

- subroutine, public [fudu::inudu](#)

Initialise the UDUNITS2 library and read the units database.

- subroutine, public [fudu::slot](#) (us1, us2, slope, offset)

Slope and offset for unit conversion.

- subroutine [fudu::uduerr](#) (udfun, unit, unit2)

Variables

- integer(c_int), parameter `fudu::ascii` =0
- integer(kind=int64), parameter, public `fudu::cfds` =24*`cfhs`
- integer, parameter, public `fudu::cfhs` =3600
- integer(c_int), parameter `fudu::latin1` =1
- integer(c_int), parameter `fudu::ute_bad_arg` =1
- integer(c_int), parameter `fudu::ute_cant_format` =9
- integer(c_int), parameter `fudu::ute_exists` =2
- integer(c_int), parameter `fudu::ute_meaningless` =6
- integer(c_int), parameter `fudu::ute_no_second` =7
- integer(c_int), parameter `fudu::ute_no_unit` =3
- integer(c_int), parameter `fudu::ute_not_same_system` =5
- integer(c_int), parameter `fudu::ute_open_arg` =12
- integer(c_int), parameter `fudu::ute_open_default` =14
- integer(c_int), parameter `fudu::ute_open_env` =13
- integer(c_int), parameter `fudu::ute_os` =4
- integer(c_int), parameter `fudu::ute_parse` =15
- integer(c_int), parameter `fudu::ute_success` =0
- integer(c_int), parameter `fudu::ute_syntax` =10
- integer(c_int), parameter `fudu::ute_unknown` =11
- integer(c_int), parameter `fudu::ute_visit_error` =8
- integer(c_int), private `fudu::utencoding`
- integer(c_int), parameter `fudu::utf8` =2
- integer(c_intptr_t), private `fudu::utsystem`

7.19.1 Data Type Documentation

7.19.1.1 type `fudu::quantity`

Type for keeping values and units together; enables easy input of units in namelists.

Definition at line 14 of file `fudu.F90`.

Class Members

<code>character(kind=c_char, len=250)</code>	<code>unit</code>	units string
<code>real(c_double)</code>	<code>value</code>	numeric value

7.20 fudu.F90

[Go to the documentation of this file.](#)

```

00001
00006 MODULE fudu
00007   use, INTRINSIC :: iso_c_binding
00008   USE iso_fortran_env, ONLY: int64
00009   IMPLICIT NONE
00010   PRIVATE
00011   PUBLIC :: inudu, exudu, convert, slot
00014   TYPE, PUBLIC :: quantity
00015     REAL(c_double) :: value
00016     CHARACTER(KIND=C_CHAR, LEN=250) :: unit
00017   END TYPE quantity

```

```

00018 INTEGER, PARAMETER, PUBLIC :: cfhs=3600
00019 INTEGER(KIND=INT64), PARAMETER, PUBLIC :: cfds=24*cfhs
00020 INTEGER(C_INT), PARAMETER :: ascii=0, latin1=1, utf8=2, ute_success=0, ute_bad_arg=1,&
00021     ute_exists=2, ute_no_unit=3, ute_os=4, ute_not_same_system=5, ute_meaningless=6,&
00022     ute_no_second=7, ute_visit_error=8, ute_cant_format=9, ute_syntax=10, ute_unknown=11,&
00023     ute_open_arg=12, ute_open_env=13, ute_open_default=14, ute_parse=15
00024 INTEGER(C_INTPTR_T), PRIVATE :: utsystem
00025 INTEGER(C_INT), PRIVATE :: utencoding
00026 INTERFACE
00027
00028     INTEGER(C_INTPTR_T) FUNCTION ut_read_xml (FILE) BIND(C)
00029     use, INTRINSIC :: iso_c_binding
00030     IMPLICIT NONE
00031     CHARACTER(KIND=C_CHAR) :: file
00032 END FUNCTION ut_read_xml
00033
00034     INTEGER(C_INTPTR_T) FUNCTION ut_parse (UTSYS, US, UTENC) BIND(C)
00035     use, INTRINSIC :: iso_c_binding
00036     IMPLICIT NONE
00037     INTEGER(C_INTPTR_T), VALUE :: utsys
00038     CHARACTER(KIND=C_CHAR) :: us
00039     INTEGER(C_INT), VALUE :: utenc
00040 END FUNCTION ut_parse
00041
00042     REAL(c_double) function ut_encode_time (year, month, day, hour, minute,&
00043     second) BIND(C)
00044     use, INTRINSIC :: iso_c_binding
00045     IMPLICIT NONE
00046     INTEGER(C_INT), VALUE :: year, month, day, hour, minute
00047     REAL(c_double), VALUE :: second
00048 END FUNCTION ut_encode_time
00049
00049     INTEGER(C_INTPTR_T) FUNCTION ut_get_converter (UP1, UP2) BIND(C)
00050     use, INTRINSIC :: iso_c_binding
00051     IMPLICIT NONE
00052     INTEGER(C_INTPTR_T), VALUE :: up1, up2
00053 END FUNCTION ut_get_converter
00054
00054     INTEGER(C_INT) FUNCTION ut_get_status () BIND(C)
00055     use, INTRINSIC :: iso_c_binding
00056     IMPLICIT NONE
00057 END FUNCTION ut_get_status
00058
00058     SUBROUTINE ut_decode_time (time, year, month, day, hour, minute, second, res) BIND(C)
00059     use, INTRINSIC :: iso_c_binding
00060     IMPLICIT NONE
00061     REAL(C_DOUBLE), VALUE :: time
00062     INTEGER(C_INT) :: year, month, day, hour, minute
00063     REAL(C_DOUBLE) :: second, res
00064 END SUBROUTINE ut_decode_time
00065
00065     SUBROUTINE cv_free (CVCNV) BIND(C)
00066     use, INTRINSIC :: iso_c_binding
00067     IMPLICIT NONE
00068     INTEGER(C_INTPTR_T), VALUE :: CVCNV
00069 END SUBROUTINE cv_free
00070
00070     SUBROUTINE ut_free (UP) BIND(C)
00071     use, INTRINSIC :: iso_c_binding
00072     IMPLICIT NONE
00073     INTEGER(C_INTPTR_T), VALUE :: UP
00074 END SUBROUTINE ut_free
00075
00075     SUBROUTINE ut_free_system (UTSYS) BIND(C)
00076     use, INTRINSIC :: iso_c_binding
00077     IMPLICIT NONE
00078     INTEGER(C_INTPTR_T), VALUE :: UTSYS
00079 END SUBROUTINE ut_free_system
00080
00080 END INTERFACE
00081
00081     INTERFACE cv_convert
00082     REAL(c_float) function cv_convert_float (cvcnv, val) BIND(C)
00083     use, INTRINSIC :: iso_c_binding
00084     IMPLICIT NONE
00085     INTEGER(C_INTPTR_T), VALUE :: cvcnv
00086     REAL(c_float), VALUE :: val
00087 END FUNCTION cv_convert_float
00088
00088     REAL(c_double) function cv_convert_double (cvcnv, val) BIND(C)
00089     use, INTRINSIC :: iso_c_binding
00090     IMPLICIT NONE
00091     INTEGER(C_INTPTR_T), VALUE :: cvcnv
00092     REAL(c_double), VALUE :: val
00093 END FUNCTION cv_convert_double
00094 END INTERFACE cv_convert
00095
00095 CONTAINS
00096
00096     SUBROUTINE inudu
00097     IMPLICIT NONE
00098     CHARACTER(LEN=*) , PARAMETER :: ududb=uduxml
00099     utencoding = utf8
00100 #ifdef GFORTTRAN
00101     utsystem = ut_read_xml(ududb//char(0))
00102 #else
00103     utsystem = ut_read_xml(ududb)
00104 #endif
00105     IF (utsystem.EQ.0) CALL uduerr ('UT_READ_XML', ududb)
00106 END SUBROUTINE inudu

```

```

00108 SUBROUTINE exudu
00109     IMPLICIT NONE
00110     CALL ut_free_system (utsystem)
00111 END SUBROUTINE exudu
00113 FUNCTION convert (qty, unit) RESULT (val)
00114     IMPLICIT NONE
00115     TYPE(quantity), INTENT(IN) :: qty
00116     CHARACTER(KIND=C_CHAR,LEN=*), INTENT(IN) :: unit
00117     INTEGER(C_INTPTR_T) :: up1, up2, cvcnv
00118     REAL(c_double) :: val
00119     up1 = ut_parse(utsystem, trim(qty%unit)//c_null_char, utencoding)
00120     IF (up1.EQ.0) CALL uduerr ('UT_PARSE(convert)', qty%unit)
00121     up2 = ut_parse(utsystem, trim(unit)//c_null_char, utencoding)
00122     IF (up2.EQ.0) CALL uduerr ('UT_PARSE(convert)', unit)
00123     cvcnv = ut_get_converter(up1, up2)
00124     IF (ut_get_status().NE.0) CALL uduerr ('UT_GET_CONVERTER(convert)', &
00125     qty%unit, unit)
00126     val = cv_convert(cvcnv, qty%value)
00127     CALL cv_free (cvcnv)
00128     CALL ut_free (up1)
00129     CALL ut_free (up2)
00130 END FUNCTION convert
00138 SUBROUTINE slot (us1, us2, slope, offset)
00139     IMPLICIT NONE
00140     CHARACTER(KIND=C_CHAR,LEN=*), INTENT(IN) :: us1, us2
00141     REAL(c_double), INTENT(OUT) :: slope
00142     REAL(c_double), INTENT(OUT), OPTIONAL :: offset
00143     INTEGER(C_INTPTR_T) :: up1, up2, cvcnv
00144     up1 = ut_parse(utsystem, trim(us1)//c_null_char, utencoding)
00145     IF (up1.EQ.0) CALL uduerr ('UT_PARSE(slot)', us1)
00146     up2 = ut_parse(utsystem, trim(us2)//c_null_char, utencoding)
00147     IF (up2.EQ.0) CALL uduerr ('UT_PARSE(slot)', us2)
00148     cvcnv = ut_get_converter(up1, up2)
00149     IF (ut_get_status().NE.0) CALL uduerr ('UT_GET_CONVERTER(slot)', &
00150     us1, us2)
00151     slope = cv_convert(cvcnv, real(1, c_double))
00152     IF (PRESENT(offset)) offset = cv_convert(cvcnv, real(0, c_double))
00153     CALL cv_free (cvcnv)
00154     CALL ut_free (up1)
00155     CALL ut_free (up2)
00156 END SUBROUTINE slot
00157 SUBROUTINE uduerr (udfun, unit, unit2)
00158     IMPLICIT NONE
00159     CHARACTER(LEN=*), DIMENSION(15), PARAMETER :: ut_errmsg=(/ character(len=18) :: 'UT_BAD_ARG', &
00160     'UT_EXISTS', 'UT_NO_UNIT', 'UT_OS', 'UT_NOT_SAME_SYSTEM', 'UT_MEANINGLESS', &
00161     'UT_NO_SECOND', 'UT_VISIT_ERROR', 'UT_CANT_FORMAT', 'UT_SYNTAX', 'UT_UNKNOWN', &
00162     'UT_OPEN_ARG', 'UT_OPEN_ENV', 'UT_OPEN_DEFAULT', 'UT_PARSE'/)
00163     CHARACTER(LEN=*), INTENT(IN) :: udfun
00164     CHARACTER(LEN=*), INTENT(IN), OPTIONAL :: unit, unit2
00165     IF (PRESENT(unit2)) THEN
00166         WRITE (0, '(UDUNITS:"A,": "A," for "A," and "A,")') trim(udfun), &
00167         trim(ut_errmsg(ut_get_status())), trim(unit), trim(unit2)
00168     ELSEIF (PRESENT(unit)) THEN
00169         WRITE (0, '(UDUNITS:"A,": "A," for "A,")') trim(udfun), &
00170         trim(ut_errmsg(ut_get_status())), trim(unit)
00171     ELSE
00172         WRITE (0, '(UDUNITS:"A,": "A,")') trim(udfun), &
00173         trim(ut_errmsg(ut_get_status()))
00174     END IF
00175     stop
00176 END SUBROUTINE uduerr
00177 END MODULE fudu

```

7.21 /Users/mpahlow/oppla/src/julian.f90 File Reference

Data Types

- interface `julian::c_strftime`
Interface to POSIX `strftime`. Returns a formatted time string, given input time struct and format. See <https://www.cplusplus.com/reference/ctime/strftime> for reference.
- interface `julian::c_strptime`
Interface to POSIX `strptime`. Returns a time struct object based on the input time string `str` and format. See <https://man7.org/linux/man-pages/man3/strptime.3.html> for reference.
- type `julian::leap`
- type `julian::tm_struct`
Derived type for compatibility with C and C++ struct `tm`. [More...](#)
- type `julian::yms`

Modules

- module `julian`
Fortran implementation of the `julian library`.

Functions/Subroutines

- subroutine, public `julian::jul_dhmsofsec` (secs, day, hour, minute, seconds)
convert number of seconds to days, hours, minutes and seconds
- subroutine `julian::jul_dsofsec` (secs, day, seconds)
- integer function `julian::jul_dutcofynd` (year, month, day)
- subroutine `julian::jul_fixym` (year, month)
- character(len=200) function, public `julian::jul_formatdate` (dutc, seconds, fmt)
Format a date and time according to a format string.
- subroutine `julian::jul_gregymdofjd` (jd, year, month, day)
- subroutine `julian::jul_initleaps`
- logical function `julian::jul_isleapday` (dutc)
Determines whether a given day contains a leap second.
- integer function `julian::jul_jdofgregymd` (year, month, day)
- integer function `julian::jul_jdofjulymd` (year, month, day)
- subroutine `julian::jul_julymdofjd` (jd, year, month, day)
- integer function `julian::jul_leapsecs` (dutc)
number of leap seconds elapsed before a given day
- integer function `julian::jul_leapsecsym` (year, month)
- subroutine, public `julian::jul_parsedt` (string, fmt, dutc, secs)
Interprets a character string as a date and time.
- real(dp) function `julian::jul_taiofdutc` (dutc)
This internal function calculates the number of seconds TAI from the floating-point number of days UTC relative to noon J2000.
- real(dp) function `julian::jul_taiofet` (et)
Convert from ET (ephemeris time) to TAI (atomic time).
- real(dp) function `julian::jul_taiofjd` (jd, type)
Convert Julian date to a number of seconds relative to J2000 TAI.
- subroutine, public `julian::jul_ydofdutc` (dutc, year, doy)
Returns the calendar year and day-of-year number for a day number relative to January 1, 2000 for a given date.
- subroutine, public `julian::jul_ymdofdutc` (dutc, year, month, day)
Returns the calendar year, month and day for a day number relative to January 1, 2000 for a given date.
- integer function `julian::leap_index` (year, month)

Variables

- integer, parameter `julian::dp` =SELECTED_REAL_KIND(KIND(0D0))
- integer, parameter `julian::gregorian_day` =15
- integer, parameter `julian::gregorian_dutc` =-152384
- integer, parameter `julian::gregorian_month` =10
- integer, parameter `julian::gregorian_year` =1582
- real(dp), parameter `julian::jd_of_j2000_noon` =2451545.0_dp
- integer, parameter `julian::jdn_of_j2000` =2451545
Julian day numer of 1 January 2000, i.e., since 4713-01-01 BCE = -4712-01-01.
- integer, parameter `julian::jul_et_type` =2
- integer, parameter `julian::jul_tai_type` =1

- integer, parameter `julian::jul_utc_type` =0
- `type(yms)`, `dimension(*)`, parameter `julian::leap_defaults` =(/ `yms(1972, 1, 10)`, `yms(1972, 7, 11)`, `yms(1973, 1, 12)`, `yms(1974, 1, 13)`, `yms(1975, 1, 14)`, `yms(1976, 1, 15)`, `yms(1977, 1, 16)`, `yms(1978, 1, 17)`, `yms(1979, 1, 18)`, `yms(1980, 1, 19)`, `yms(1981, 7, 20)`, `yms(1982, 7, 21)`, `yms(1983, 7, 22)`, `yms(1985, 7, 23)`, `yms(1988, 1, 24)`, `yms(1990, 1, 25)`, `yms(1991, 1, 26)`, `yms(1992, 7, 27)`, `yms(1993, 7, 28)`, `yms(1994, 7, 29)`, `yms(1996, 1, 30)`, `yms(1997, 7, 31)`, `yms(1999, 1, 32)`, `yms(2006, 1, 33)`, `yms(2009, 1, 34)`, `yms(2012, 7, 35)`, `yms(2015, 7, 36)`, `yms(2017, 1, 37)`/)
- integer, `dimension(:)`, allocatable `julian::leap_table`
leap-seconds table
- `type(leap)` `julian::leaps`
- `real(dp)`, parameter `julian::mjd_of_j2000_noon` =51544.5_dp

7.21.1 Data Type Documentation

7.21.1.1 type `julian::leap`

Definition at line 23 of file `julian.f90`.

Class Members

integer	nmax	number of leap seconds as of ymax
integer	nmin	number of leap second before ymin = 1972
integer	size	size of leap-seconds table
integer	ymax	last year of leap-seconds table
integer	ymin	first year of leap-seconds table (1972)

7.21.1.2 type `julian::tm_struct`

Derived type for compatibility with C and C++ struct `tm`.

Enables calling `strftime` and `strptime` using `iso_c_binding`.
[com/reference/ctime/tm](https://www.cplusplus.com/reference/ctime/tm) for reference.

See <https://www.cplusplus.com/reference/ctime/tm>

Definition at line 35 of file `julian.f90`.

Class Members

integer(c_int)	<code>tm_hour</code> = 0	Hours [0–23].
integer(c_int)	<code>tm_isdst</code> = 0	DST [–1/0/1].
integer(c_int)	<code>tm_mday</code> = 0	Day [1–31].
integer(c_int)	<code>tm_min</code> = 0	Minutes [0–59].
integer(c_int)	<code>tm_mon</code> = 0	Month [0–11].
integer(c_int)	<code>tm_sec</code> = 0	Seconds [0–60] (1 leap second)
integer(c_int)	<code>tm_wday</code> = 0	Day of week [0–6].
integer(c_int)	<code>tm_yday</code> = 0	Days in year [0–365].
integer(c_int)	<code>tm_year</code> = 0	Year – 1900.

7.21.1.3 type julian::yms

Definition at line 18 of file [julian.f90](#).

Class Members

integer	month	month for leap second
integer	seconds	cumulative leap seconds at month, year
integer	year	year for leap second

7.22 julian.f90

[Go to the documentation of this file.](#)

```

00001
00007 MODULE julian
00008   use, INTRINSIC :: iso_c_binding, only: c_char, c_int, c_null_char
00009   IMPLICIT NONE
00010   PRIVATE
00011   PUBLIC :: jul_dhmsotsec, jul_formatdate, jul_parsedt, jul_ydofdutc, jul_ymdofdutc
00012   INTEGER, PARAMETER :: dp=selected_real_kind(kind(0d0)), &
00013       jul_utc_type=0, jul_tai_type=1, jul_et_type=2, &
00014       jdn_of_j2000=2451545, &
00015       gregorian_year=1582, &
00016       gregorian_month=10, gregorian_day=15, gregorian_dutc=-152384
00017   REAL(dp), PARAMETER :: mjd_of_j2000_noon=51544.5_dp, jd_of_j2000_noon=2451545.0_dp
00018   TYPE :: yms
00019       INTEGER :: year, &                !< year for leap second
00020       month, &                          !< month for leap second
00021       seconds
00022   END TYPE yms
00023   TYPE :: leap
00024       INTEGER :: nmin, &                !< number of leap second before ymin = 1972
00025       nmax, &                          !< number of leap seconds as of ymax
00026       ymin, &                          !< first year of leap-seconds table (1972)
00027       ymax, &                          !< last year of leap-seconds table
00028       size
00029   END TYPE leap
00030
00035   TYPE, BIND(C) :: tm_struct
00036       INTEGER(C_INT) :: tm_sec = 0
00037       INTEGER(C_INT) :: tm_min = 0
00038       INTEGER(C_INT) :: tm_hour = 0
00039       INTEGER(C_INT) :: tm_mday = 0
00040       INTEGER(C_INT) :: tm_mon = 0
00041       INTEGER(C_INT) :: tm_year = 0
00042       INTEGER(C_INT) :: tm_wday = 0
00043       INTEGER(C_INT) :: tm_yday = 0
00044       INTEGER(C_INT) :: tm_isdst = 0
00045   END TYPE tm_struct
00046   TYPE(yms), DIMENSION(*), PARAMETER :: leap_defaults=(/ &
00047       yms(1972, 1, 10), &
00048       yms(1972, 7, 11), &
00049       yms(1973, 1, 12), &
00050       yms(1974, 1, 13), &
00051       yms(1975, 1, 14), &
00052       yms(1976, 1, 15), &
00053       yms(1977, 1, 16), &
00054       yms(1978, 1, 17), &
00055       yms(1979, 1, 18), &
00056       yms(1980, 1, 19), &
00057       yms(1981, 7, 20), &
00058       yms(1982, 7, 21), &
00059       yms(1983, 7, 22), &
00060       yms(1985, 7, 23), &
00061       yms(1988, 1, 24), &
00062       yms(1990, 1, 25), &
00063       yms(1991, 1, 26), &
00064       yms(1992, 7, 27), &
00065       yms(1993, 7, 28), &
00066       yms(1994, 7, 29), &
00067       yms(1996, 1, 30), &
00068       yms(1997, 7, 31), &
00069       yms(1999, 1, 32), &

```



```

00070      yms(2006, 1, 33), &
00071      yms(2009, 1, 34), &
00072      yms(2012, 7, 35), &
00073      yms(2015, 7, 36), &
00074      yms(2017, 1, 37))
00075  TYPE(leap) :: leaps
00076  INTEGER, ALLOCATABLE :: leap_table(:)
00077  INTERFACE
00078
00081      FUNCTION c_strftime (str, slen, form, tm) BIND (C, NAME='strftime') RESULT (rc)
00082      IMPORT :: c_char, c_int, tm_struct
00083      CHARACTER(KIND=C_CHAR), INTENT(OUT) :: str(*)
00084      INTEGER(C_INT), VALUE, INTENT(IN) :: slen
00085      CHARACTER(KIND=C_CHAR), INTENT(IN) :: form(*)
00086      TYPE(tm_struct), INTENT(IN) :: tm
00087      INTEGER(C_INT) :: rc
00088  END FUNCTION c_strftime
00092  FUNCTION c_strptime (str, format, tm) BIND (C, NAME='strptime') RESULT (rc)
00093  IMPORT :: c_char, c_int, tm_struct
00094  CHARACTER(KIND=C_CHAR), INTENT(IN) :: str(*)
00095  CHARACTER(KIND=C_CHAR), INTENT(IN) :: format(*)
00096  TYPE(tm_struct), INTENT(INOUT) :: tm
00097  CHARACTER(KIND=C_CHAR, LEN=1) :: rc
00098  END FUNCTION c_strptime
00099  END INTERFACE
00100  CONTAINS
00113  FUNCTION jul_taiofjd (jd, type) RESULT (tai)
00114  IMPLICIT NONE
00115  REAL(dp), INTENT(IN) :: jd
00116  INTEGER, INTENT(IN) :: type
00118  REAL(dp) :: et, tai
00119  SELECT CASE (type)
00120  CASE (jul_tai_type)
00121      tai = (jd - jd_of_j2000_noon)*86400._dp
00122  CASE (jul_et_type)
00123      et = (jd - jd_of_j2000_noon)*86400._dp
00124      tai = jul_taiofet(et)
00125  CASE DEFAULT
00126      tai = jul_taiofdutc(jd - jd_of_j2000_noon)
00127  END SELECT
00128  END FUNCTION jul_taiofjd
00136  FUNCTION jul_taiofet (et) RESULT (tai)
00137  IMPLICIT NONE
00138  REAL(dp), PARAMETER :: m0=6.239996_dp, m1=1.99096871e-7_dp, eb=1.671e-2_dp, &
00139      k=1.657e-3_dp, delta_t_a=32.184_dp
00140  REAL(dp), INTENT(IN) :: et
00141  REAL(dp) :: tai, mean_anom, ecc_anom
00142  mean_anom = m0 + m1*et
00143  ecc_anom = mean_anom + eb*sin(mean_anom)
00144  tai = et - delta_t_a - k*sin(ecc_anom)
00145  END FUNCTION jul_taiofet
00150  FUNCTION jul_taiofdutc (dutc) RESULT (tai)
00151  IMPLICIT NONE
00152  REAL(dp), INTENT(IN) :: dutc
00153  REAL(dp) :: tai, frac, leaps
00154  INTEGER :: dutc_floor, dutc_noon
00155  dutc_floor = floor(dutc)
00156  dutc_noon = dutc_floor
00157  leaps = jul_leapsecs(dutc_noon)
00158  frac = dutc - dutc_floor
00159  IF (frac.GT.0._dp) frac = frac*(86400._dp + jul_leapsecs(dutc_noon + 1) - leaps)
00160  tai = 86400._dp*dutc_noon + leaps + frac
00161  END FUNCTION jul_taiofdutc
00163  FUNCTION jul_leapsecs (dutc) RESULT (seconds)
00164  IMPLICIT NONE
00165  INTEGER, INTENT(IN) :: dutc
00166  INTEGER :: seconds, year, month, day
00167  CALL jul_ymdofdutc (dutc, year, month, day)
00168  seconds = jul_leapsecsym(year, month)
00169  END FUNCTION jul_leapsecs
00170  FUNCTION leap_index (year, month) RESULT (index)
00171  IMPLICIT NONE
00172  INTEGER, INTENT(IN) :: year, month
00173  INTEGER :: index
00174  index = 2*(year - leaps%ymmin) + (month - 1)/6 + 1
00175  END FUNCTION leap_index
00176  FUNCTION jul_leapsecsym (year, month) RESULT (seconds)
00177  IMPLICIT NONE
00178  INTEGER, INTENT(IN) :: year, month
00179  INTEGER :: seconds, index
00180  IF (.NOT.ALLOCATED(leap_table)) CALL jul_initleaps
00181  index = leap_index(year, month)
00182  IF (index < 0) THEN
00183      seconds = leaps%nmin
00184  ELSE IF (index >= leaps%size) THEN
00185      seconds = leaps%nmax
00186  ELSE

```

```

00187         seconds = leap_table(index)
00188     END IF
00189 END FUNCTION jul_leapsecsym
00191 FUNCTION jul_isleapday (dutc) RESULT (isLeapDay)
00192     IMPLICIT NONE
00193     INTEGER, INTENT(IN) :: dutc
00194     LOGICAL :: isleapday
00195     INTEGER :: year, month, day
00196     IF (.NOT.ALLOCATED(leap_table)) CALL jul_initleaps
00197     CALL jul_ymdofdutc (dutc, year, month, day)
00198     isleapday = .false.
00199     IF (day.LE.29) RETURN
00200     IF (day.EQ.30) THEN
00201         IF (month.NE.6) RETURN
00202     ELSEIF (day.EQ.31) THEN
00203         IF (month.NE.12) RETURN
00204     ELSE
00205         RETURN ! invalid date
00206     END IF
00207     isleapday = jul_leapsecsym(year, month + 1).NE.jul_leapsecsym(year, month)
00208 END FUNCTION jul_isleapday
00209 FUNCTION jul_dutcofynd (year, month, day) RESULT (dutc)
00210     IMPLICIT NONE
00211     INTEGER, INTENT(OUT) :: year, month
00212     INTEGER, INTENT(IN) :: day
00213     INTEGER :: dutc
00214     CALL jul_fixym (year, month)
00215     IF ((year.GT.gregorian_year).OR.((year.EQ.gregorian_year)&
00216         .AND.((month.GT.gregorian_month)&
00217         .OR.((month.EQ.gregorian_month).AND.(day.GE.gregorian_day)))) THEN
00218         dutc = jul_jdofgregymd(year, month, day) - jdn_of_j2000
00219     ELSE
00220         dutc = jul_jdofjulymd(year, month, day) - jdn_of_j2000
00221     END IF
00222 END FUNCTION jul_dutcofynd
00223 FUNCTION jul_jdofgregymd (year, month, day) RESULT (jd)
00224     IMPLICIT NONE
00225     INTEGER, INTENT(IN) :: year, month, day
00226     INTEGER :: jd, tmp
00227     tmp = 0
00228     IF (month.LE.2) tmp = -1
00229     jd = (1461*(year + 4800 + tmp))/4 + (367*(month - 2 - 12*tmp))/12 &
00230         - (3*((year + 4900 + tmp)/100))/4 + day - 32075
00231 END FUNCTION jul_jdofgregymd
00232 FUNCTION jul_jdofjulymd (year, month, day) RESULT (jd)
00233     IMPLICIT NONE
00234     INTEGER, INTENT(IN) :: year, month, day
00235     INTEGER :: jd
00236     jd = 367*year - (7*(year + 5001 + (month-9)/7))/4 + (275*month)/9 &
00237         + day + 1729777
00238 END FUNCTION jul_jdofjulymd
00244 SUBROUTINE jul_ydofdutc(dutc, year, doy)
00245     IMPLICIT NONE
00246     INTEGER, INTENT(IN) :: dutc
00247     INTEGER, INTENT(OUT) :: year
00248     INTEGER, INTENT(OUT) :: doy
00249     INTEGER :: month, day, jan=1
00250     CALL jul_ymdofdutc(dutc, year, month, day)
00251     doy = dutc - jul_dutcofynd(year, jan, 1) + 1
00252 END SUBROUTINE jul_ydofdutc
00258 SUBROUTINE jul_ymdofdutc (dutc, year, month, day)
00259     IMPLICIT NONE
00260     INTEGER, INTENT(IN) :: dutc
00261     INTEGER, INTENT(OUT) :: year
00262     INTEGER, INTENT(OUT) :: month
00263     INTEGER, INTENT(OUT) :: day
00264     IF (dutc >= gregorian_dutc) THEN
00265         CALL jul_gregymdofjd (dutc + jdn_of_j2000, year, month, day)
00266     ELSE
00267         CALL jul_julymdofjd (dutc + jdn_of_j2000, year, month, day)
00268     END IF
00269 END SUBROUTINE jul_ymdofdutc
00270 SUBROUTINE jul_gregymdofjd(jd, year, month, day)
00271     IMPLICIT NONE
00272     INTEGER, INTENT(IN) :: jd
00273     INTEGER, INTENT(OUT) :: year, month, day
00274     INTEGER :: l, n, i, j
00275     l = jd + 68569
00276     n = (4*l)/146097
00277     l = l - (146097*n + 3)/4
00278     i = (4000*(l+1))/1461001
00279     l = l - (1461*i)/4 + 31
00280     j = (80*l)/2447
00281     day = l - (2447*j)/80
00282     l = j/11
00283     month = j + 2 - 12*l
00284     year = 100*(n-49) + i + l

```

```

00285 END SUBROUTINE jul_gregymdofjd
00286 SUBROUTINE jul_julymdofjd (jd, year, month, day)
00287   IMPLICIT NONE
00288   INTEGER, INTENT(IN) :: jd
00289   INTEGER, INTENT(OUT) :: year, month, day
00290   INTEGER :: l, n, i, j, k
00291   j = jd + 1402;
00292   k = (j-1)/1461;
00293   l = j - 1461*k;
00294   n = (l-1)/365 - 1/1461;
00295   i = l - 365*n + 30;
00296   j = (80*i)/2447;
00297   day = i - (2447*j)/80;
00298   i = j/11;
00299   month = j + 2 - 12*i;
00300   year = 4*k + n + i - 4716;
00301 END SUBROUTINE jul_julymdofjd
00302 SUBROUTINE jul_fixym (year, month)
00303   IMPLICIT NONE
00304   INTEGER, INTENT(INOUT) :: year, month
00305   INTEGER :: delta_year
00306   DO WHILE (month < 1)
00307     month = month + 12
00308     year = year - 1
00309   END DO
00310   delta_year = (month - 1) / 12
00311   month = month - 12 * delta_year
00312 END SUBROUTINE jul_fixym
00313 SUBROUTINE jul_initleaps
00314   IMPLICIT NONE
00315   INTEGER :: n, nsecs
00316   IF (ALLOCATED(leap_table)) RETURN
00317   leaps%ymmin = leap_defaults(1)%year
00318   leaps%nmmin = leap_defaults(1)%seconds - 1
00319   leaps%ymax = leap_defaults(SIZE(leap_defaults))%year
00320   leaps%nmmax = leap_defaults(SIZE(leap_defaults))%seconds
00321   leaps%size = 2 * (leaps%ymax - leaps%ymmin + 1) + 1
00322   ALLOCATE (leap_table(leaps%size))
00323   leap_table = leaps%nmmin
00324   DO n=1,SIZE(leap_defaults)
00325     leap_table(leap_index(leap_defaults(n)%year, leap_defaults(n)%month)) =
00326     leap_defaults(n)%seconds
00327   END DO
00328   nsecs = leaps%nmmin
00329   DO n=1,SIZE(leap_defaults)
00330     IF (leap_table(n) == leaps%nmmin) THEN
00331       leap_table(n) = nsecs
00332     ELSE
00333       IF (leap_table(n).NE.nsecs + 1) THEN
00334         WRITE (*, '( "Illegal leap second step from ",I0," to ",I0)') nsecs, leap_table(n)
00335         stop
00336       END IF
00337       nsecs = leap_table(n)
00338     END IF
00339   END DO
00340 END SUBROUTINE jul_initleaps
00341 SUBROUTINE jul_dhmsfsec (secs, day, hour, minute, seconds)
00342   IMPLICIT NONE
00343   INTEGER, INTENT(IN) :: secs
00344   INTEGER, INTENT(OUT) :: day,& !< number of days
00345   hour,& !< number of hours into day (0--23)
00346   minute,& !< number of minutes into hour (0--59)
00347   seconds
00348   CALL jul_dsofsec(secs, day, seconds)
00349   hour = floor(seconds / 3600._dp)
00350   seconds = seconds - 3600*hour
00351   minute = floor(seconds / 60._dp)
00352   seconds = seconds - 60*minute
00353 END SUBROUTINE jul_dhmsfsec
00354 ! convert number of seconds to days, plus seconds into day
00355 SUBROUTINE jul_dsofsec (secs, day, seconds)
00356   IMPLICIT NONE
00357   INTEGER, INTENT(IN) :: secs
00358   INTEGER, INTENT(OUT) :: day, seconds
00359   day = floor(secs / 86400._dp)
00360   seconds = secs - 86400*day
00361   IF (seconds.LT.0) THEN
00362     day = day - 1
00363     seconds = seconds + 86400
00364   END IF
00365 END SUBROUTINE jul_dsofsec
00366 SUBROUTINE jul_parsedt (string, fmt, dutc, secs)
00367   IMPLICIT NONE
00368   CHARACTER(LEN=*), INTENT(IN) :: string,& !< the character string to parse
00369   fmt
00370   INTEGER, INTENT(OUT) :: dutc,& !< inferred days relative to January 1, 2000
00371   secs

```

```

00380     TYPE(tm_struct) :: tm
00381     CHARACTER(KIND=C_CHAR, LEN=1) :: status
00382     INTEGER :: year, month
00383     tm = tm_struct()
00384     status = c_strptime(trim(string)//c_null_char, format=trim(fmt)//c_null_char, tm=tm)
00385     year = tm%tm_year + 1900
00386     month = tm%tm_mon + 1
00387     dutc = jul_dutcofynd(year, month, tm%tm_mday)
00388     secs = tm%tm_sec + tm%tm_min*60 + tm%tm_hour*3600
00389 END SUBROUTINE jul_parsedt
00396 FUNCTION jul_formatdate (dutc, seconds, fmt) RESULT (string)
00397     IMPLICIT NONE
00398     INTEGER, INTENT(IN) :: dutc, seconds
00399     CHARACTER(LEN=*), INTENT(IN) :: fmt
00400     CHARACTER(LEN=200) :: string
00401     INTEGER :: len, year, month, day
00402     TYPE(tm_struct) :: tm
00403     tm = tm_struct()
00404     CALL jul_ymdofdutc (dutc, year, month, day)
00405     tm%tm_year = year - 1900
00406     tm%tm_mon = month - 1
00407     tm%tm_mday = day
00408     tm%tm_hour = seconds / 3600
00409     tm%tm_min = (seconds - 3600 * tm%tm_hour) / 60
00410     tm%tm_sec = seconds - 3600 * tm%tm_hour - 60 * tm%tm_min
00411     len = c_strftime(str=string, slen=200, form=trim(fmt)//c_null_char, tm=tm)
00412     string = string(1:len)
00413 END FUNCTION jul_formatdate
00414 END MODULE julian

```

7.23 /Users/mpahlow/oppla/src/lambert.f90 File Reference

Data Types

- interface [lambert::lambertw](#)

Modules

- module [lambert](#)

F90 module for the Lambert-W function made from TOMS 743.

Functions/Subroutines

- elemental real([dp](#)) function [lambert::lwm1](#) (x)
lwm1: closed-form approximation of -1-branch of the Lambert-W function
- elemental double precision function [lambert::wapd](#) (x, nb, l)
- elemental real function [lambert::wapr](#) (x, nb, l)

Variables

- integer, parameter, private [lambert::dp](#) =KIND(OD0)

7.24 lambert.f90

[Go to the documentation of this file.](#)

```

00001
00004 MODULE lambert
00005   IMPLICIT NONE
00006   INTEGER, PARAMETER, PRIVATE :: dp=kind(0d0)
00007   INTERFACE lambertw
00008     MODULE PROCEDURE wapr, wapr
00009   END INTERFACE
00010 CONTAINS
00012   ! after Barry et al. (2000)
00013   ! the maximum relative error of this approximation is 0.025%
00014   ELEMENTAL FUNCTION lwml (x) RESULT (w)
00015     IMPLICIT NONE
00016     REAL(dp), PARAMETER :: m1=0.3361d0, m2=-0.0042d0, m3=-0.0201d0
00017     REAL(dp), INTENT(IN) :: x
00018     REAL(dp) :: w, s
00019     s = -1d0 - log(-x)
00020     w = -1d0 - s - 2d0/m1*(1d0 - 1d0/(1d0 + (m1*sqrt(0.5d0*s)&
00021                                     / (1d0 + m2*s*exp(m3*sqrt(s))))))
00022   END FUNCTION lwml
00023 !
00024 !
00025 ! Approximating the W function
00026 !
00027 !
00028   ELEMENTAL FUNCTION wapr (X, NB, L) RESULT (WAP)
00029 !
00030 ! WAPR - output
00031 ! X - argument of W(X)
00032 ! NB is the branch of the W function needed:
00033 ! NB = 0 - upper branch
00034 ! NB <> 0 - lower branch
00035 !
00036 ! NERROR is the output error flag:
00037 ! NERROR = 0 -> routine completed successfully
00038 ! NERROR = 1 -> X is out of range
00039 !
00040 ! Range: -exp(-1) <= X for the upper branch of the W function
00041 ! -exp(-1) < X < 0 for the lower branch of the W function
00042 !
00043 ! L - determines how WAPR is to treat the argument X
00044 ! L = 1 -> X is the offset from -exp(-1), so compute
00045 ! W(X-exp(-1))
00046 ! L <> 1 -> X is the desired X, so compute W(X)
00047 !
00048 ! M - print messages from WAPR?
00049 ! M = 1 -> Yes
00050 ! M <> 1 -> No
00051 !
00052 ! ++++++
00053 !
00054 ! NN is the output device number
00055 !
00056 ! NBITS is the number of bits (less 1) in the mantissa of the
00057 ! floating point number representation of your machine.
00058 ! It is used to determine the level of accuracy to which the W
00059 ! function should be calculated.
00060 !
00061 ! Most machines use a 24-bit matissa for single precision and
00062 ! 53-56 bits for double precision. The IEEE standard is 53
00063 ! bits. The Fujitsu VP2200 uses 56 bits. Long word length
00064 ! machines vary, e.g., the Cray X/MP has a 48-bit mantissa for
00065 ! single precision.
00066 !
00067   IMPLICIT NONE
00068   INTEGER, PARAMETER :: nn=6, nbits=23, niter=1
00069   REAL, PARAMETER :: em=-0.367879441171442,& ! -EXP(-1)
00070                     em9=-1.234098040866796e-4,& ! -EXP(-9)
00071                     c13=1.e0/3.e0,&
00072                     c23=2.e0*c13,&
00073                     em2=2.e0/em,&
00074                     e12=-em2,&
00075                     tb=.5e0**nbits,&
00076                     tb2=.5e0**(nbits/2),& ! Sqrt(TB)
00077                     x0=0.0350769390096679055,& ! TB**(1/6)*0.5E0
00078                     x1=0.302120119432784731,& ! (1 - 17*TB**(2/7))*EM
00079                     an22=3.6e2/83.e0,&
00080                     an11=135./83.e0,&
00081                     an3=8.e0/3.e0,&
00082                     an4=135.e0/83.e0,&
00083                     an5=166.e0/39.e0,&
00084                     an6=3167.e0/3549.e0,&
00085                     s2=1.41421356237310,& ! Sqrt(2.E0)

```

```

00086             s21=2.e0*s2-3.e0,&
00087             s22=4.e0-3.e0*s2,&
00088             s23=s2-2.e0
00089 REAL :: x, wap, an2, delx, xx, reta, z1, t, ts, eta, temp, temp2, zn
00090 INTEGER :: nb, 1, null
00091 INTENT(IN) :: x, nb, 1
00092 null = 0
00093 !      Various mathematical constants
00094
00095 !
00096 !      The following COMMON statement is needed only when testing this
00097 !      function using BISECT, otherwise it can be removed.
00098 !
00099 !      COMMON/WAPCOM/NBITS
00100 !      DATA INIT,NITER/0,1/
00101 !      DATA NBITS/23/
00102 !
00103 !      IF (INIT.EQ.0) THEN
00104 !          INIT=1
00105 !
00106 !      Code to calculate NBITS for the host machine. NBITS is the
00107 !      mantissa length less one. This value is chosen to allow for
00108 !      rounding error in the final bit. This need only be run once on
00109 !      any particular machine. It can then be included in the above
00110 !      DATA statement.
00111 !
00112 !      DO I=1,2000
00113 !          B=2.E0**(-I)
00114 !          V=1.E0+B
00115 !          IF (V.EQ.1.E0) THEN
00116 !              NBITS=I-1
00117 !              J=-ALOG10(B)
00118 !              IF (M.EQ.1) WRITE (NN,40) NBITS, J
00119 !              EXIT
00120 !          ENDIF
00121 !      END DO
00122 !
00123 !      Remove to here after NBITS has been calculated once
00124 !
00125 !      The case of large NBITS
00126 !
00127 !      IF (NBITS.GE.56) NITER=2
00128 !
00129 !      Various mathematical constants
00130 !
00131 !      EM=-EXP(-1.E0)
00132 !      EM9=-EXP(-9.E0)
00133 !      C13=1.E0/3.E0
00134 !      C23=2.E0*C13
00135 !      EM2=2.E0/EM
00136 !      E12=-EM2
00137 !      TB=.5E0*NBITS
00138 !      TB2=SQRT(TB)
00139 !      X0=TB*(1.E0/6.E0)*.5E0
00140 !      X1=(1.E0-17.E0*TB*(2.E0/7.E0))*EM
00141 !      AN22=3.6E2/83.E0
00142 !      AN11=135./83.E0
00143 !      AN3=8.E0/3.E0
00144 !      AN4=135.E0/83.E0
00145 !      AN5=166.E0/39.E0
00146 !      AN6=3167.E0/3549.E0
00147 !      S2=SQRT(2.E0)
00148 !      S21=2.E0*S2-3.E0
00149 !      S22=4.E0-3.E0*S2
00150 !      S23=S2-2.E0
00151 !      ENDIF
00152 !      IF (1.EQ.1) THEN
00153 !          delx=x
00154 !          IF (delx.LT.0.e0) THEN
00155 !              wap = 1./null
00156 !              RETURN
00157 !          END IF
00158 !          xx=x+em
00159 !          IF (E12*DELX.LT.TB*.2.AND.M.EQ.1) WRITE (NN,60) DELX
00160 !      ELSE
00161 !          IF (x.LT.em) THEN
00162 !              wap = 1./null
00163 !              RETURN
00164 !          END IF
00165 !          IF (x.EQ.em) THEN
00166 !              wap=-1.e0
00167 !              RETURN
00168 !          ENDIF
00169 !          xx=x
00170 !          delx=xx-em
00171 !          IF (DELX.LT.TB2.AND.M.EQ.1) WRITE (NN,70) XX
00172 !      ENDIF

```

```

00173      IF (nb.EQ.0) THEN
00174      !
00175      !      Calculations for Wp
00176      !
00177      IF (abs(xx).LE.x0) THEN
00178          wap=xx/(1.e0+xx/(1.e0+xx/(2.e0+xx/(.6e0+.34e0*xx))))
00179          RETURN
00180      ELSE IF (xx.LE.x1) THEN
00181          reta=sqrt(e12*delx)
00182          wap=reta/(1.e0+reta/(3.e0+reta/(reta/(an4+reta/(reta*&
00183              an6+an5))+an3)))-1.e0
00184          RETURN
00185      ELSE IF (xx.LE.2.e1) THEN
00186          reta=s2*sqrt(1.e0-xx/em)
00187          an2=4.612634277343749e0*sqrt(sqrt(reta+&
00188              1.09556884765625e0))
00189          wap=reta/(1.e0+reta/(3.e0+(s21*an2+s22)*reta/&
00190              (s23*(an2+reta)))-1.e0
00191      ELSE
00192          z1=alog(xx)
00193          wap=alog(xx/alog(xx/z1**exp(-1.124491989777808e0/&
00194              (.4225028202459761e0+z1))))
00195      ENDIF
00196      ELSE
00197      !
00198      !      Calculations for Wm
00199      !
00200      IF (xx.GE.0.e0) THEN
00201          wap = -1./null
00202          RETURN
00203      END IF
00204      IF (xx.LE.x1) THEN
00205          reta=sqrt(e12*delx)
00206          wap=reta/(reta/(3.e0+reta/(reta/(an4+reta/(reta*&
00207              an6-an5))-an3))-1.e0)-1.e0
00208          RETURN
00209      ELSE IF (xx.LE.em9) THEN
00210          z1=alog(-xx)
00211          t=-1.e0-z1
00212          ts=sqrt(t)
00213          wap=z1-(2.e0*ts)/(s2+(c13-t/(2.7e2+&
00214              ts*127.0471381349219e0))*ts)
00215      ELSE
00216          z1=alog(-xx)
00217          eta=2.e0-em2*xx
00218          wap=alog(xx/alog(-xx/((1.e0-.5043921323068457e0*&
00219              (z1+1.e0))*(sqrt(eta)+eta/3.e0)+1.e0)))
00220      ENDIF
00221      ENDIF
00222      !      DO I=1,NITER
00223      !      zn=alog(xx/wap)-wap
00224      !      temp=1.e0+wap
00225      !      temp2=temp+c23*zn
00226      !      temp2=2.e0*temp*temp2
00227      !      wap=wap*(1.e0+(zn/temp)*(temp2-zn)/(temp2-2.e0*zn))
00228      !      END DO
00229      RETURN
00230      ! 40 FORMAT(/,' NBITS is',I4,'.',/, ' Expect',I4,&
00231      !      ' significant digits from WAPR.')
00232      ! 50 FORMAT(/,' Warning: the offset x is negative (it must be > 0)')
00233      ! 60 FORMAT(' Warning: For this offset (' ,E16.8,')',/,&
00234      !      ' W is negligibly different from -1')
00235      ! 70 FORMAT(' Warning: x (= ',E16.8,') is close to -exp(-1).',/,&
00236      !      ' Enter x as an offset to -exp(-1) for greater accuracy')
00237      !
00238      !      END of WAPR
00239      END FUNCTION wapr
00240      !
00241      !
00242      !
00243      !      Approximating the W function
00244      !
00245      !
00246      ELEMENTAL FUNCTION wapd(X,NB,L) RESULT (WAP)
00247      !
00248      !      WAPR - output
00249      !      X - argument of W(X)
00250      !      NB is the branch of the W function needed:
00251      !      NB = 0 - upper branch
00252      !      NB <> 0 - lower branch
00253      !
00254      !      NERROR is the output error flag:
00255      !      NERROR = 0 -> routine completed successfully
00256      !      NERROR = 1 -> X is out of range
00257      !
00258      !      Range: -exp(-1) <= X for the upper branch of the W function
00259      !      -exp(-1) < X < 0 for the lower branch of the W function

```

```

00260 !
00261 ! L - determines how WAPR is to treat the argument X
00262 ! L = 1 -> X is the offset from -exp(-1), so compute
00263 ! W(X-exp(-1))
00264 ! L <> 1 -> X is desired X, so compute W(X)
00265 !
00266 ! M - print messages from WAPR?
00267 ! M = 1 -> Yes
00268 ! M <> 1 -> No
00269 !
00270 ! ++++++
00271 !
00272 ! NN is the output device number
00273 !
00274 ! NBITS is the number of bits (less 1) in the mantissa of the
00275 ! floating point number representation of your machine.
00276 ! It is used to determine the level of accuracy to which the W
00277 ! function should be calculated.
00278 !
00279 ! Most machines use a 24-bit mantissa for single precision and
00280 ! 53-56 bits for double precision. The IEEE standard is 53
00281 ! bits. The Fujitsu VP2200 uses 56 bits. Long word length
00282 ! machines vary, e.g., the Cray X/MP has a 48-bit mantissa for
00283 ! single precision.
00284 !
00285 ! IMPLICIT NONE
00286 ! INTEGER, PARAMETER :: nn=6, nbits=52
00287 ! Various mathematical constants
00288 ! DOUBLE PRECISION, PARAMETER :: em=-0.367879441171442d0,& ! -EXP(-1.D0)
00289 ! em9=-1.234098040866796d-4,& ! -EXP(-9.D0)
00290 ! c13=1.d0/3.d0,&
00291 ! c23=2.d0*c13,&
00292 ! em2=2.d0/em,&
00293 ! d12=-em2,&
00294 ! tb=.5d0*nbits,&
00295 ! tb2=.5d0**(nbits/2),& ! SQRT(TB)
00296 ! x0=1.23039165028796206d-3,& ! TB**(1/6)*.5
00297 ! x1=0.367668719700312234d0,& ! (1 - 17*TB**(2/7))*EM
00298 ! an3=8.d0/3.d0,&
00299 ! an4=135.d0/83.d0,&
00300 ! an5=166.d0/39.d0,&
00301 ! an6=3167.d0/3549.d0,&
00302 ! s2=1.41421356237310d0,& ! SQRT(2.D0)
00303 ! s21=2.d0*s2 - 3.d0,&
00304 ! s22=4.d0 - 3.d0*s2,&
00305 ! s23=s2 - 2.d0
00306 ! DOUBLE PRECISION :: x, wap, an2, delx, xx, reta, ts, zl, zn, t, eta, temp, temp2
00307 ! INTEGER :: nb, l, null
00308 ! INTENT(IN) :: x, nb, l
00309 ! null = 0
00310 !
00311 ! The following COMMON statement is needed only when testing this
00312 ! function using BISECT, otherwise it can be removed.
00313 !
00314 ! COMMON/WAPCOM/NBITS
00315 ! DATA INIT,NITER/0,1/
00316 ! DATA NBITS/52/
00317 !
00318 ! IF(INIT.EQ.0) THEN
00319 ! INIT=1
00320 !
00321 ! Code to calculate NBITS for the host machine. NBITS is the
00322 ! mantissa length less one. This value is chosen to allow for
00323 ! rounding error in the final bit. This need only be run once on
00324 ! any particular machine. It can then be included in the above
00325 ! DATA statement.
00326 !
00327 ! DO I=1,2000
00328 ! B=2.D0**(-I)
00329 ! V=1.D0+B
00330 ! IF(V.EQ.1.D0) THEN
00331 ! NBITS=I-1
00332 ! J=-DLOG10(B)
00333 ! IF(M.EQ.1) WRITE(NN,40)NBITS,J
00334 ! EXIT
00335 ! ENDIF
00336 ! END DO
00337 !
00338 ! Remove to here after NBITS has been calculated once
00339 !
00340 ! The case of large NBITS
00341 !
00342 ! IF(NBITS.GE.56) NITER=2
00343 !
00344 ! Various mathematical constants
00345 !
00346 ! EM=-EXP(-1.D0)

```



```

00347 !      EM9=-EXP (-9.D0)
00348 !      C13=1.D0/3.D0
00349 !      C23=2.D0*C13
00350 !      EM2=2.D0/EM
00351 !      D12=-EM2
00352 !      TB=.5D0*NBITS
00353 !      TB2=SQRT (TB)
00354 !      X0=TB** (1.D0/6.D0) *.5D0
00355 !      X1=(1.D0-17.D0*TB** (2.D0/7.D0) ) *EM
00356 !      AN3=8.D0/3.D0
00357 !      AN4=135.D0/83.D0
00358 !      AN5=166.D0/39.D0
00359 !      AN6=3167.D0/3549.D0
00360 !      S2=SQRT (2.D0)
00361 !      S21=2.D0*S2-3.D0
00362 !      S22=4.D0-3.D0*S2
00363 !      S23=S2-2.D0
00364 !      ENDIF
00365 !      IF (1.EQ.1) THEN
00366 !      delx=x
00367 !      IF (delx.LT.0.d0) THEN
00368 !      wap = 1d0/null
00369 !      RETURN
00370 !      END IF
00371 !      xx=x+em
00372 !      IF (D12*DELX.LT.TB**2.AND.M.EQ.1) WRITE (NN,60) DELX
00373 !      ELSE
00374 !      IF (x.LT.em) THEN
00375 !      wap = 1d0/null
00376 !      RETURN
00377 !      END IF
00378 !      IF (x.EQ.em) THEN
00379 !      wap=-1.d0
00380 !      RETURN
00381 !      ENDIF
00382 !      xx=x
00383 !      delx=xx-em
00384 !      IF (DELX.LT.TB2.AND.M.EQ.1) WRITE (NN,70) XX
00385 !      ENDIF
00386 !      IF (nb.EQ.0) THEN
00387 !      !
00388 !      !      Calculations for Wp
00389 !      !
00390 !      IF (abs (xx) .LE.x0) THEN
00391 !      wap=xx/ (1.d0+xx/ (1.d0+xx/ (2.d0+xx/ (.6d0+.34d0*xx) ) ) )
00392 !      RETURN
00393 !      ELSE IF (xx.LE.x1) THEN
00394 !      reta=dsqrt (d12*delx)
00395 !      wap=reta/ (1.d0+reta/ (3.d0+reta/ (reta/ (an4+reta/ (reta*an6+an5) ) &
00396 !      +an3) ) )-1.d0
00397 !      RETURN
00398 !      ELSE IF (xx.LE.2.d1) THEN
00399 !      reta=s2*dsqrt (1.d0-xx/em)
00400 !      an2=4.612634277343749d0*dsqrt (dsqrt (reta+1.09556884765625d0) )
00401 !      wap=reta/ (1.d0+reta/ (3.d0+ (s21*an2+s22) *reta/ (s23* (an2+reta) ) ) )-1.d0
00402 !      ELSE
00403 !      z1=dlog (xx)
00404 !      wap=dlog (xx/dlog (xx/z1**dexp (-1.124491989777808d0/ (.4225028202459761d0+z1) ) ) )
00405 !      ENDIF
00406 !      ELSE
00407 !      !
00408 !      !      Calculations for Wm
00409 !      !
00410 !      IF (xx.GE.0.d0) THEN
00411 !      wap = -1d0/null
00412 !      RETURN
00413 !      END IF
00414 !      IF (xx.LE.x1) THEN
00415 !      reta=dsqrt (d12*delx)
00416 !      wap=reta/ (reta/ (3.d0+reta/ (reta/ (an4+reta/ (reta*an6+an5) ) )-an3) )-1.d0)-1.d0
00417 !      RETURN
00418 !      ELSE IF (xx.LE.em9) THEN
00419 !      z1=dlog (-xx)
00420 !      t=-1.d0-z1
00421 !      ts=dsqrt (t)
00422 !      wap=z1- (2.d0*ts) / (s2+ (c13-t/ (2.7d2+ts*127.0471381349219d0) ) *ts)
00423 !      ELSE
00424 !      z1=dlog (-xx)
00425 !      eta=2.d0-em2*xx
00426 !      wap=dlog (xx/dlog (-xx/ ( (1.d0-.5043921323068457d0* (z1+1.d0) ) &
00427 !      * (dsqrt (eta) +eta/3.d0) +1.d0) ) )
00428 !      ENDIF
00429 !      ENDIF
00430 !      DO I=1,NITER
00431 !      zn=dlog (xx/wap) -wap
00432 !      temp=1.d0+wap
00433 !      temp2=temp+c23*zn

```

```

00434      temp2=2.d0*temp*temp2
00435      wap=wap*(1.d0+(zn/temp)*(temp2-zn)/(temp2-2.d0*zn))
00436 !      END DO
00437      RETURN
00438 ! 40  FORMAT(/,' NBITS is',I4,'.',/, ' Expect',I4,' significant digits from WAPD.')
00439 ! 50  FORMAT(/,' Warning: the offset x is negative (it must be > 0)')
00440 ! 60  FORMAT(' Warning: For this offset ('',D16.8,''),',/, ' W is negligibly different from -1')
00441 ! 70  FORMAT(' Warning: x (= '',D16.8,'') is close to -exp(-1).',/, &
00442 !      ' Enter x as an offset to -exp(-1) for greater accuracy')
00443 !
00444 !      END of WAPD
00445      END FUNCTION wapd
00446 END MODULE lambert

```

7.25 /Users/mpahlow/oppla/src/onf.f90 File Reference

Data Types

- type [onf::dataset](#)
- type [onf::dimension](#)
- type [onf::nf90grd](#)
- type [onf::nf90group](#)
- type [onf::nf90stratt](#)
- type [onf::nf90var](#)
- type [onf::nf90vec](#)
- type [onf::phydat](#)
physical data
- interface [onf::rpd](#)
- type [onf::statevars](#)

Modules

- module [onf](#)
NetCDF interface for oppla.

Functions/Subroutines

- subroutine [onf::exof](#) (stv)
Close output file.
- subroutine [onf::innf](#) (states, phys, lun, atts)
Initialise NETCDF interface and read associated parameters.
- subroutine [onf::inpd](#) (pdf, times, env, box, svd)
Open physical data file and read forcing datasets.
- subroutine [onf::invar](#) (stv, ofn, control, dfn, pfn, force, resume)
Read initial state-variable values and create and initialize output file.
- subroutine [onf::nf90info](#) (group, latdeg, londeg, t0, t1, writeable)
open groupfilename, read global attributes and coordinates (depth, lon, lat, time)
- subroutine [onf::onf_error](#) (caller, nffun, name, nferr)
- subroutine [onf::outsv](#) (stv, times, env)
Write time in UTC and current state-variable values to file.
- subroutine [onf::read_dim](#) (dim, ncid, caller, nferr)
Load dimension variable.
- subroutine [onf::read_ds](#) (ds, group, data)
read dataset in group described by ds into array data
- subroutine [onf::readpd](#) (pda, time)
Read physical data for time interval encompassing phytim.
- subroutine [set_att](#) (group, ds, natt)
- subroutine [onf::write_output](#) (stv, env, times)

Variables

- character(len=14), parameter, public [onf::bfan](#) ='Bottom File'
- character(len=14), parameter, public [onf::dfan](#) ='Data File'
- integer, public [onf::ico2ds](#)
- character(len=14), parameter, public [onf::infan](#) ='Start File'
- integer, public [onf::iprsds](#)
- character(len=14), parameter, public [onf::latan](#) ='latitude'
- character(len=14), parameter, public [onf::lonan](#) ='longitude'
- integer [onf::nbvds](#)
- integer [onf::ngdds](#)
- integer [onf::npfds](#)
- integer [onf::nsfds](#)
- integer [onf::nsvds](#)
- real([dp](#)), [dimension](#)(:,:), allocatable [onf::offsets](#)
- real([dp](#)), [dimension](#)(:,:,:), allocatable, target [onf::pdgds](#)
- real([dp](#)), [dimension](#)(:,:), allocatable, target [onf::pdsds](#)
- character(len=14), parameter, public [onf::pfan](#) ='Parameter File'
- character(len=14), parameter, public [onf::plfan](#) ='Plankton File'
- character(len=14), parameter, public [onf::sufan](#) ='Spin-up File'
- character(len=14), parameter, public [onf::timan](#) ='Time'
- character(len=nf90_max_name), public [onf::timunt](#) ='s'

7.25.1 Data Type Documentation

7.25.1.1 type [onf::dataset](#)

Definition at line 9 of file [onf.f90](#).

Class Members

character(len=nf90_max_name)	name ="	
character(len=nf90_max_name)	type ="	data type, e.g., Temperature, Salinity, etc.

7.25.1.2 type [onf::nf90stratt](#)

Definition at line 13 of file [onf.f90](#).

Class Members

integer	id	
character(len=nf90_max_name)	name	
character(len=nf90_max_name)	value	

7.25.2 Function/Subroutine Documentation

7.25.2.1 set_att()

```
subroutine inpd::set_att (
    class(nf90group), intent(inout) group,
    class(nf90var), intent(in) ds,
    integer, intent(in) natt ) [private]
```

Definition at line 714 of file [onf.f90](#).

Referenced by [onf::inpd\(\)](#).

Here is the caller graph for this function:



7.26 onf.f90

[Go to the documentation of this file.](#)

```
00001
00002
00003 MODULE onf
00004   USE netcdf
00005   USE et
00006   IMPLICIT NONE
00007   PRIVATE
00008   PUBLIC :: nf90_max_name
00009   TYPE :: dataset
00010       CHARACTER(LEN=NF90_MAX_NAME) :: type=",& !< data type, e.g., Temperature, Salinity, etc.
00011       name=""
00012   END TYPE dataset
00013   TYPE :: nf90stratt
00014       INTEGER :: id
00015       CHARACTER(LEN=NF90_MAX_NAME) :: name, value
00016   END TYPE nf90stratt
00017   TYPE :: nf90var
00018       CHARACTER(LEN=NF90_MAX_NAME) :: name=",&
00019       type=",& !< data type, e.g., Temperature, Salinity, etc.
00020       units=""
00021       INTEGER :: varid,& !< variable ID
00022       ndims
00023   CONTAINS
00024       PROCEDURE :: read => read_ds
00025   END TYPE nf90var
00026   TYPE, EXTENDS(nf90var) :: nf90grd
00027       REAL(kind(0d0)), DIMENSION(:,:), POINTER :: values
00028   END TYPE nf90grd
00029   TYPE, EXTENDS(nf90var) :: nf90vec
00030       REAL(kind(0d0)), DIMENSION(:), POINTER :: values
00031   END TYPE nf90vec
00032   TYPE :: dimension
00033       INTEGER :: id=0,& !< dimension ID
00034       varid
00035       CHARACTER(LEN=NF90_MAX_NAME) :: name=",& units=""
00036       REAL(dp), ALLOCATABLE :: values(:)
00037       TYPE(nf90grd) :: bounds
00038   CONTAINS
00039       PROCEDURE :: read => read_dim
00040   END TYPE dimension
```

```

00041 TYPE :: nf90group
00042   CHARACTER(LEN=NF90_MAX_NAME) :: filename
00043   INTEGER :: id=0, it0, ilat, ilon, ntim, nbox=0,&
00044     timediff=0
00045   TYPE(dimension) :: depth, depth_flux, time, lat, lon, state, bounds
00046   TYPE(nf90stratt), ALLOCATABLE :: attributes(:)
00047   CONTAINS
00048     PROCEDURE :: info => nf90info
00049 END TYPE nf90group
00050 TYPE, EXTENDS(nf90group), PUBLIC :: phydat
00051   INTEGER, DIMENSION(:), ALLOCATABLE :: isflx, ibot
00052   TYPE(nf90vec), DIMENSION(:), ALLOCATABLE :: surface, bottom, surflx, profile
00053   TYPE(nf90grd), DIMENSION(:), ALLOCATABLE :: data
00054   CHARACTER(LEN=NF90_MAX_NAME) :: advect='upwind'
00055   REAL(dp), DIMENSION(:), POINTER :: surflux, botbc, surpar
00056   REAL(dp), ALLOCATABLE :: sds(:), pds(:,,:), cds(:,:)
00057   REAL(dp) :: fpar=0.43_dp
00058   LOGICAL :: clsbot=.false.
00059   PROCEDURE(rpd), POINTER :: get => nopd
00060   CONTAINS
00061     PROCEDURE :: init => inpd
00062 END TYPE phydat
00063 TYPE, EXTENDS(nf90group), PUBLIC :: statevars
00064   INTEGER :: nsv_one_box,&      !< No. of plankton state variables in each layer
00065     nsv_all_boxes,&            !< total number of plankton state variables (across all layers)
00066     nsv_total,&                !< total number of all (including diagnostic) state variables
00067     nfl,&                      !< No. of flux levels in output
00068     i0flux,&                  !< starting index of fluxes in state-variable vector
00069     i0sms,&                   !< starting index of soma in state-variable vector
00070     nsms
00071   INTEGER, POINTER :: it(:)
00072   INTEGER, DIMENSION(:), ALLOCATABLE :: start, count
00073   LOGICAL :: write_soma=.false.
00074   LOGICAL, POINTER :: smsk(:,:)
00075   CHARACTER(LEN=NF90_MAX_NAME) :: svgn=""
00076   REAL(dp), DIMENSION(:,,:), POINTER :: data,& !< array of state variables (pointer to all_states)
00077     roc
00078   REAL(dp), DIMENSION(:), ALLOCATABLE :: all_states, flux_depth
00079   TYPE(nf90vec), DIMENSION(:), ALLOCATABLE :: var
00080   TYPE(nf90vec) :: flux, soma
00081   PROCEDURE(write_states), POINTER :: write_output => write_states
00082   CONTAINS
00083     PROCEDURE :: open => innf, init => invar, write => outsv, close => exof!, read => readsv
00084 END TYPE statevars
00085 INTEGER :: nbvds, nsvds, nsfds, ngdds, npfds
00086 CHARACTER(LEN=14), PARAMETER, PUBLIC :: pfan='Parameter File',&
00087   plfan='Plankton File', dfan='Data File', infan='Start File',&
00088   bfan='Bottom File', sufan='Spin-up File', timan='Time',&
00089   latan='latitude', lonan='longitude'
00090 CHARACTER(LEN=NF90_MAX_NAME), PUBLIC :: timunt='s'
00091 INTEGER, PUBLIC :: iprsds, ico2ds
00092 REAL(dp), ALLOCATABLE :: offsets(:,:)
00093 REAL(dp), ALLOCATABLE, TARGET :: pdsds(:,,:), pdgds(:,,:)
00094 abstract INTERFACE
00095   SUBROUTINE rpd (pdf, time)
00096     IMPORT phydat, dp
00097     IMPLICIT NONE
00098     CLASS(phydat), INTENT(INOUT) :: pdf
00099     REAL(dp), INTENT(IN) :: time
00100   END SUBROUTINE rpd
00101 END INTERFACE
00102 CONTAINS
00103 SUBROUTINE innf (states, phys, lun,atts)
00104   IMPLICIT NONE
00105   CLASS(statevars), INTENT(OUT) :: states
00106   TYPE(phydat), INTENT(OUT) :: phys
00107   INTEGER, INTENT(IN) :: lun
00108   TYPE(datasets), DIMENSION(100) :: forcing, profile, bottom, surface, surflx
00109   LOGICAL :: clsbot, soma
00110   CHARACTER(LEN=NF90_MAX_NAME) :: time, depth, lat, lon, bounds, iomsg, svgroup, advect
00111   REAL(dp) :: fpar
00112   TYPE(nf90stratt), DIMENSION(:), ALLOCATABLE, INTENT(OUT), OPTIONAL :: atts
00113   namelist /start/ time, depth, lat, lon, bounds, svgroup, soma
00114   namelist /physics/ time, depth, lat, lon, bounds, forcing, profile, bottom, surface, surflx,&
00115     clsbot, advect, fpar
00116   time = 'Time'
00117   depth = 'Depth'
00118   lat = 'latitude'
00119   lon = 'longitude'
00120   bounds = 'bounds'
00121   soma = .false.
00122   svgroup = ""
00123   rewind(lun)
00124   READ (lun, nml=start, END=900, ERR=900, IOMSG=iomsg)
00125   IF (soma) WRITE (stderr,("innf: Ignoring soma in namelist start. Please use namelist fluxes!"))
00126   states%time%name = trim(time)
00127 END SUBROUTINE innf

```

```

00131     states%depth%name = trim(depth)
00132     states%lat%name = trim(lat)
00133     states%lon%name = trim(lon)
00134     states%bounds%name = trim(bounds)
00135     states%svgn = trim(svggroup)
00136     time = 'Time'
00137     depth = 'Depth'
00138     lat = 'latitude'
00139     lon = 'longitude'
00140     bounds = 'bounds'
00141     advect = phys%advect
00142     clsbot = phys%clsbot
00143     fpar = phys%fPAR
00144     rewind(lun)
00145     READ (lun, nml=physics, END=900, ERR=900, IOMSG=iomsg)
00146     phys%time%name = trim(time)
00147     phys%depth%name = trim(depth)
00148     phys%lat%name = trim(lat)
00149     phys%lon%name = trim(lon)
00150     phys%bounds%name = trim(bounds)
00151     phys%fPAR = fpar
00152     phys%advect = advect
00153     phys%clsbot = clsbot
00154     ngdds = count(forcing%type.NE.") ! No. of gridded physical forcings
00155     npfds = count(profile%type.NE.") ! No. of temporally constant vertical profiles
00156     nbvds = count(bottom%type.NE.") ! number of bottom value datasets
00157     nsvds = count(surface%type.NE.") ! number of surface value datasets
00158     nsfds = count(surflx%type.NE.") ! number of surface flux datasets
00159     ALLOCATE (phys%data(ngdds), phys%profile(npfds), phys%bottom(nbvds), phys%surface(nsvds), &
00160              phys%surflx(nsfds))
00161     phys%data%name = forcing(1:ngdds)%name
00162     phys%data%type = forcing(1:ngdds)%type
00163     phys%bottom%name = bottom(1:nbvds)%name
00164     phys%bottom%type = bottom(1:nbvds)%type
00165     phys%surface%name = surface(1:nsvds)%name
00166     phys%surface%type = surface(1:nsvds)%type
00167     phys%surflx%name = surflx(1:nsfds)%name
00168     phys%surflx%type = surflx(1:nsfds)%type
00169     phys%profile%name = profile(1:npfds)%name
00170     phys%profile%type = profile(1:npfds)%type
00171     IF (PRESENT(atts)) THEN
00172         ALLOCATE (atts(6))
00173         atts%name = (/dfan, infan, pfan, plfan, bfan, sufan/) ! attribute names
00174     END IF
00175     RETURN
00176 900 CONTINUE
00177     WRITE (stderr, '( "innf: ", A )') trim(iomsg)
00178     stop
00179 END SUBROUTINE innf
00180
00192 SUBROUTINE nf90info (group, latdeg, londeg, t0, t1, writeable)
00193     IMPLICIT NONE
00194     CLASS(nf90group), INTENT(INOUT) :: group
00195     REAL(dp), INTENT(INOUT), OPTIONAL :: latdeg, londeg
00196     REAL(dp), INTENT(INOUT), OPTIONAL :: t0
00197     REAL(dp), INTENT(INOUT), OPTIONAL :: t1
00198     LOGICAL, INTENT(IN), OPTIONAL :: writeable
00199     INTEGER :: n, nt, natts, nferr, i(1), rank, mode
00200     REAL(dp), ALLOCATABLE :: edges(:)
00201     IF (group%id.EQ.0) THEN
00202         mode = nf90_nowrite
00203         IF (PRESENT(writeable)) THEN
00204             IF (writeable) mode = nf90_write
00205         END IF
00206         nferr = nf90_open(path=trim(group%filename), mode=mode, ncid=group%id)
00207         IF (nferr.NE.nf90_noerr) CALL onf_error ('nf90info', 'NF90_OPEN', trim(group%filename), nferr)
00208     END IF
00209     nferr = nf90_inquire(ncid=group%id, nattributes=natts)
00210     nferr = nf90_inq_dimid(ncid=group%id, name=group%time%name, dimid=group%time%id)
00211     IF (nferr.NE.nf90_noerr) CALL onf_error ('nf90info', 'NF90_INQ_DIMID', group%time%name, nferr)
00212     CALL group%time%read(ncid=group%id, caller='nf90info')
00213     nferr = nf90_inq_dimid(ncid=group%id, name=group%lat%name, dimid=group%lat%id)
00214     IF (nferr.EQ.nf90_noerr) THEN ! latitude stored as a dataset
00215         CALL group%lat%read(ncid=group%id, caller='nf90info')
00216     ELSEIF (len_trim(group%lat%name).GT.0) THEN ! latitude stored as an attribute
00217         ALLOCATE (group%lat%values(1))
00218         nferr = nf90_get_att(ncid=group%id, varid=nf90_global, name=group%lat%name, &
00219                          values=group%lat%values)
00220         IF (nferr.NE.nf90_noerr) CALL onf_error ('nf90info', 'NF90_GET_ATT', group%lat%name, nferr)
00221     ELSE
00222         ALLOCATE (group%lat%values(0))
00223     END IF
00224     IF (PRESENT(latdeg).AND.(SIZE(group%lat%values).GT.0)) THEN
00225         i = minloc(abs(group%lat%values - latdeg))
00226         group%ilat = i(1)
00227         IF (abs(group%lat%values(group%ilat) - latdeg).GT.1d-6) WRITE (stdout,100) &
00228             trim(group%lat%name), latdeg, trim(group%filename), group%lat%values(group%ilat)

```

```

00229      ! use actual lat/lon from forcing for light cycle and output file
00230      latdeg = group%lat%values(group%ilat)
00231      ELSE
00232        group%ilat = 0
00233      END IF
00234      nferr = nf90_inq_dimid(ncid=group%id, name=group%lon$name, dimid=group%lon%id)
00235      IF (nferr.EQ.nf90_noerr) THEN
00236        CALL group%lon%read (ncid=group%id, caller='nf90info')
00237      ELSEIF (len_trim(group%lon$name).GT.0) THEN
00238        ALLOCATE (group%lon%values(1))
00239        nferr = nf90_get_att(ncid=group%id, varid=nf90_global, name=group%lon$name, &
00240          values=group%lon%values)
00241        IF (nferr.NE.nf90_noerr) CALL onf_error ('nf90info', 'NF90_GET_ATT', group%lon$name, nferr)
00242      ELSE
00243        ALLOCATE (group%lon%values(0))
00244      END IF
00245      IF (PRESENT(londeg).AND.(SIZE(group%lon%values).GT.0)) THEN
00246        i = minloc(abs(group%lon%values - londeg))
00247        group%ilon = i(1)
00248        IF (abs(group%lon%values(group%ilon) - londeg).GT.1d-6) WRITE (stdout,100) &
00249          trim(group%lon$name), londeg, trim(group%filename), group%lon%values(group%ilon)
00250        ! use actual lat/lon from forcing for light cycle and output file
00251        londeg = group%lon%values(group%ilon)
00252      ELSE
00253        group%ilon = 0
00254      END IF
00255      nferr = nf90_inq_dimid(ncid=group%id, name=group%bounds$name, dimid=group%bounds%id)
00256      IF (nferr.NE.nf90_noerr) CALL onf_error ('nf90info', 'NF90_INQ_DIMID', group%bounds$name, nferr)
00257      nferr = nf90_inq_dimid(ncid=group%id, name=group%depth$name, dimid=group%depth%id)
00258      IF (nferr.NE.nf90_noerr) CALL onf_error ('nf90info', 'NF90_INQ_DIMID', group%depth$name, nferr)
00259      CALL group%depth%read (ncid=group%id, caller='nf90info')
00260      group%nbox = SIZE(group%depth%values)
00261      IF (len_trim(group%depth%bounds$name).EQ.0) THEN
00262        ALLOCATE (group%depth%bounds%values(2,group%nbox), edges(0:group%nbox))
00263        group%depth%bounds$name = trim(group%depth$name)//'_edges'
00264        group%depth%bounds%units = group%depth%units
00265        edges(0) = 0._dp
00266        DO n=1,group%nbox
00267          edges(n) = 2._dp*group%depth%values(n) - edges(n-1)
00268        END DO
00269        group%depth%bounds%values(1,:) = edges(0:group%nbox-1)
00270        group%depth%bounds%values(2,:) = edges(1:group%nbox)
00271        DEALLOCATE (edges)
00272      END IF
00273      nt = SIZE(group%time%values)
00274      IF (mode.EQ.nf90_write) THEN ! only for appending to previous output
00275        group%it0 = nt
00276        group%ntim = 1
00277      ELSEIF (PRESENT(t0).AND.(nt.GT.1)) THEN
00278        IF (t0.LE.group%time%values(1)) t0 = t0 + group%time%values(1)
00279        i = maxloc(group%time%values, mask=group%time%values.LE.t0)
00280        group%it0 = i(1)
00281        IF (PRESENT(t1)) THEN
00282          t1 = min(t1, group%time%values(nt))
00283          i = minloc(group%time%values(group%it0:), mask=group%time%values(group%it0:).GE.t1)
00284          group%ntim = i(1)
00285        ELSE
00286          group%ntim = 1
00287        END IF
00288      ELSE
00289        group%it0 = 1
00290        group%ntim = nt
00291      END IF
00292      group%time%values(1:group%ntim) = group%time%values(group%it0:group%it0+group%ntim-1) &
00293        + group%timediff ! convert UTC to local time
00294 100 FORMAT ("nf90info: closest ",a," to",g12.5,"in ",a," is",g12.5)
00295      END SUBROUTINE nf90info
00296
00305      SUBROUTINE invar (stv, ofn, control, dfn, pfn, force, resume)
00306        IMPLICIT NONE
00307        CLASS(statevars), TARGET, INTENT(INOUT) :: stv
00308        CHARACTER(LEN=*) , INTENT(IN) :: ofn, control, dfn, pfn
00309        LOGICAL, INTENT(IN) :: force, resume
00310        TYPE(dimension) :: sms
00311        CHARACTER(LEN=23) :: date
00312        CHARACTER(LEN=500) :: prefix="", prefix_ic="", prefix_dfn="", prefix_pfn=""
00313        CHARACTER(LEN=:), DIMENSION(:), ALLOCATABLE :: ftnames, from, to, fromto
00314        LOGICAL :: exists
00315        INTEGER, DIMENSION(:,:), ALLOCATABLE :: ifrom, ito ! indices in smsk
00316        INTEGER :: dimids(4), datevals(8)
00317        INTEGER :: id, n, nn, nferr, namelen=0, stringid, ndims, lunof
00318        ALLOCATE (offsets(stv%nsv_one_box,stv%nbox), stv%all_states(stv%nsv_total), stv%start(4),
00319          stv%count(4))
00319        stv%data(1:stv%nsv_one_box,1:stv%nbox) => stv%all_states(1:stv%nsv_all_boxes)
00320        offsets = 0d0
00321        id = stv%id ! save file ID
00322        IF (len_trim(stv%svgn).GT.0) THEN

```

```

00323         nferr = nf90_inq_ncid(ncid=id, name=trim(stv%svgn), grp_ncid=stv%id)
00324         IF (nferr.NE.nf90_noerr) &
00325             CALL onf_error (caller='invar', nffun='NF90_INQ_NCID', name=stv%svgn, nferr=nferr)
00326     END IF
00327     DO n=1, stv%nsv_one_box ! read initial conditions for all states
00328         stv%var(n)%values => stv%data(n,:)
00329         CALL stv%var(n)%read (group=stv, data=stv%var(n)%values)
00330     END DO
00331     stv%it => stv%start(2:2)
00332     stv%count = (/stv%inbox, 1, 1, 1/)
00333     stv%start = (/1, 0, 1, 1/)
00334     IF (resume) THEN
00335         stv%it = stv%it0
00336         stv%id = id
00337         RETURN
00338     END IF
00339     nferr = nf90_close(id)
00340     INQUIRE (file=trim(ofn), exist=exists)
00341     IF (exists) THEN
00342         IF (force.AND.(scan(trim(ofn), '*?').EQ.0)) THEN
00343             OPEN (newunit=lunof, file=trim(ofn))
00344             CLOSE (unit=lunof, status='DELETE')
00345         ELSE
00346             WRITE (stderr, '("Output file ",A," exists. Use -f to overwrite.")' trim(ofn))
00347             stop
00348         END IF
00349     END IF
00350     stv%depth%bounds$name = trim(stv%depth$name)//'_edges'
00351     stv%depth%bounds%units = stv%depth%units
00352     CALL date_and_time (values = datevals)
00353     WRITE (date, '(I4.4,"-",I2.2,"-",I2.2," ",I2.2,":",I2.2,":",I2.2,".",I3.3)') &
00354         datevals(/(n, n = 1,3), (n, n = 5,8)/)
00355     nn = 0
00356     IF (ofn(1:1).NE.'/') THEN ! if ofn is a relative path
00357         DO n=1, len_trim(ofn) ! make the paths of control, parameter, data files relative to ofn
00358             SELECT CASE (ofn(n:n))
00359                 CASE ('.')
00360                     nn = nn + 1
00361                 CASE ('/')
00362                     IF (nn.EQ.2) THEN ! this means ofn contains ../
00363                         prefix = " ! in this case the path cannot be reconstructed
00364                         EXIT
00365                     ELSEIF (nn.EQ.0) THEN
00366                         prefix = trim(prefix)//'../'
00367                     END IF
00368                 CASE DEFAULT
00369                     nn = 0
00370             END SELECT
00371         END DO
00372         IF (stv%filename(1:1).NE.'/') prefix_ic = prefix ! use prefix only for relative file names
00373         IF (dfn(1:1).NE.'/') prefix_dfn = prefix
00374         IF (pfn(1:1).NE.'/') prefix_pfn = prefix
00375     ELSE
00376         IF (control(1:1).NE.'/') WRITE (stdout,100) 'control', 'control'
00377         IF (pfn(1:1).NE.'/') WRITE (stdout,100) 'parameter', 'parameter'
00378         IF (dfn(1:1).NE.'/') WRITE (stdout,100) 'data', 'data'
00379         IF (stv%filename(1:1).NE.'/') WRITE (stdout,100) 'initial-conditions', 'initial-conditions'
00380     END IF
00381     nferr = nf90_create(trim(ofn), cmode=nf90_netcdf4, ncid=stv%id) ! create output file
00382     IF (nferr.NE.nf90_noerr) CALL onf_error ('invar', 'NF90_CREATE', trim(ofn), nferr)
00383     nferr = nf90_put_att(ncid=stv%id, varid=nf90_global, name='date', values=date)
00384     nferr = nf90_put_att(ncid=stv%id, varid=nf90_global, name='initial_conditions', &
00385         values=trim(prefix_ic)//trim(stv%filename))
00386     nferr = nf90_put_att(ncid=stv%id, varid=nf90_global, name='control_file',
00387         values=trim(prefix)//trim(control))
00388     nferr = nf90_put_att(ncid=stv%id, varid=nf90_global, name='data_file',
00389         values=trim(prefix_dfn)//trim(dfn))
00390     nferr = nf90_put_att(ncid=stv%id, varid=nf90_global, name='parameter_file',
00391         values=trim(prefix_pfn)//trim(pfn))
00392     ! create dimensions and dimension-variables
00393     nferr = nf90_def_dim(ncid=stv%id, name=stv%depth$name, len=stv%inbox, dimid=stv%depth%id)
00394     nferr = nf90_def_dim(ncid=stv%id, name=stv%bounds$name, len=2, dimid=stv%bounds%id)
00395     nferr = nf90_def_var(ncid=stv%id, name=stv%depth$name, xtype=nf90_double, &
00396         dimids=(/stv%depth%id/), varid=stv%depth%varid)
00397     nferr = nf90_put_att(ncid=stv%id, varid=stv%depth%varid, name='units',
00398         values=trim(stv%depth%units))
00399     nferr = nf90_put_att(ncid=stv%id, varid=stv%depth%varid, name='axis', values='Z')
00400     nferr = nf90_put_att(ncid=stv%id, varid=stv%depth%varid, name='positive', values='down')
00401     nferr = nf90_put_att(ncid=stv%id, varid=stv%depth%varid, name='bounds',
00402         values=stv%depth%bounds$name)
00403     nferr = nf90_def_var(ncid=stv%id, name=stv%depth%bounds$name, xtype=nf90_double, &
00404         dimids=(/stv%bounds%id, stv%depth%id/), varid=stv%depth%bounds%varid)
00405     nferr = nf90_put_att(ncid=stv%id, varid=stv%depth%bounds%varid, name='units',
00406         values=trim(stv%depth%bounds%units))
00407     nferr = nf90_def_dim(ncid=stv%id, name=stv%time$name, len=nf90_unlimited, dimid=stv%time%id)
00408     nferr = nf90_def_var(ncid=stv%id, name=stv%time$name, xtype=nf90_double, &
00409         dimids=(/stv%time%id/), varid=stv%time%varid)

```



```

00404      nferr = nf90_put_att(ncid=stvid, varid=stvt%time%varid, name='units', values=trim(stvt%time%units))
00405      nferr = nf90_put_att(ncid=stvid, varid=stvt%time%varid, name='axis', values='T')
00406      nferr = nf90_put_att(ncid=stvid, varid=stvt%time%varid, name='calendar', values='standard')
00407      IF (all(len_trim((/stvt%lon%name, stvt%lat%name/)).GT.0)) THEN
00408          nferr = nf90_def_dim(ncid=stvid, name=stvt%lat%name, len=1, dimid=stvt%lat%id)
00409          nferr = nf90_def_var(ncid=stvid, name=stvt%lat%name, xtype=nf90_double,&
00410                               dimids=(/stvt%lat%id/), varid=stvt%lat%varid)
00411          nferr = nf90_put_att(ncid=stvid, varid=stvt%lat%varid, name='units',
00412                               values=trim(stvt%lat%units))
00412          nferr = nf90_put_att(ncid=stvid, varid=stvt%lat%varid, name='axis', values='Y')
00413          nferr = nf90_def_dim(ncid=stvid, name=stvt%lon%name, len=1, dimid=stvt%lon%id)
00414          nferr = nf90_def_var(ncid=stvid, name=stvt%lon%name, xtype=nf90_double,&
00415                               dimids=(/stvt%lon%id/), varid=stvt%lon%varid)
00416          nferr = nf90_put_att(ncid=stvid, varid=stvt%lon%varid, name='units',
00417                               values=trim(stvt%lon%units))
00417          nferr = nf90_put_att(ncid=stvid, varid=stvt%lon%varid, name='axis', values='X')
00418          ndims = 4
00419      ELSE
00420          ndims = 2
00421      END IF
00422      dimids = (/stvt%depth%id, stvt%time%id, stvt%lon%id, stvt%lat%id/)
00423      DO n=1,stvt%nsv_one_box
00424          nferr = nf90_def_var(ncid=stvid, name=stvt%var(n)%name, xtype=nf90_double,&
00425                               dimids=dimids(1:ndims), varid=stvt%var(n)%varid)
00426          IF (nferr.NE.nf90_noerr) &
00427              CALL onf_error ('invar', 'NF90_DEF_VAR', trim(stvt%var(n)%name), nferr)
00428          nferr = nf90_put_att(ncid=stvid, varid=stvt%var(n)%varid, name='units',&
00429                               values=trim(stvt%var(n)%units))
00430      END DO
00431      IF (stvt%write_soma) THEN
00432          stvt%write_output => write_output
00433          stvt%flux%values => stvt%all_states(stvt%i0flux:stvt%i0sms-1)
00434          stvt%soma%values => stvt%all_states(stvt%i0sms:stvt%nsv_total)
00435          nferr = nf90_put_att(ncid=stvid, varid=nf90_global, name='soma', values=trim(stvt%soma%name))
00436          nferr = nf90_put_att(ncid=stvid, varid=nf90_global, name='flux', values=trim(stvt%flux%name))
00437          stvt%depth_flux%name = 'depth_flux'
00438          nferr = nf90_def_dim(ncid=stvid, name=stvt%depth_flux%name, len=stvt%nfl,
00439                               dimid=stvt%depth_flux%id)
00439          nferr = nf90_def_var(ncid=stvid, name=stvt%depth_flux%name, xtype=nf90_double,&
00440                               dimids=(/stvt%depth_flux%id/), varid=stvt%depth_flux%varid)
00441          nferr = nf90_put_att(ncid=stvid, varid=stvt%depth_flux%varid, name='units',
00442                               values=trim(stvt%depth_flux%units))
00442          nferr = nf90_put_att(ncid=stvid, varid=stvt%depth_flux%varid, name='axis', values='Z')
00443          nferr = nf90_put_att(ncid=stvid, varid=stvt%depth_flux%varid, name='positive', values='down')
00444          namelen = max(maxval(len_trim(stvt%var%name)), 11)*2 + 2
00445          ALLOCATE (ifrom(0:stvt%nsv_one_box, stvt%nsv_one_box), ito(0:stvt%nsv_one_box, stvt%nsv_one_box))
00446          ALLOCATE (CHARACTER(namelen)::ftnames(0:stvt%nsv_one_box),&
00447                       from(stvt%nsms), to(stvt%nsms), fromto(stvt%nsms))
00448          ito(0,:) = (/ (nn, nn=1, stvt%nsv_one_box) /) ! set up index matrices for state names
00449          ftnames(0) = 'aggregation'
00450          DO n=1, stvt%nsv_one_box
00451              ftnames(n) = stvt%var(n)%name
00452              ifrom(:,n) = (/ (nn, nn=0, stvt%nsv_one_box) /)
00453              ito(n,:) = (/ (nn, nn=1, stvt%nsv_one_box) /)
00454          END DO
00455          sms%name = 'sms'
00456          from = ftnames(pack(ifrom, mask=stvt%smask))
00457          to = ftnames(pack(ito, mask=stvt%smask))
00458          DO n=1, stvt%nsms
00459              fromto(n) = trim(from(n))//'->'//trim(to(n))
00460          END DO
00461          nferr = nf90_def_dim(ncid=stvid, name=stvt%state%name, len=stvt%nsv_one_box, dimid=stvt%state%id)
00462          nferr = nf90_def_dim(ncid=stvid, name=sms%name, len=stvt%nsms, dimid=sms%id)
00463          nferr = nf90_def_dim(ncid=stvid, name='string', len=namelen, dimid=string%id)
00464          nferr = nf90_def_var(ncid=stvid, name=stvt%state%name, xtype=nf90_char,&
00465                               dimids=(/string%id, stvt%state%id/), varid=stvt%state%varid)
00466          nferr = nf90_def_var(ncid=stvid, name=sms%name, xtype=nf90_char,&
00467                               dimids=(/string%id, sms%id/), varid=sms%varid)
00468          nferr = nf90_def_var(ncid=stvid, name=stvt%flux%name, xtype=nf90_double,&
00469                               dimids=(/stvt%state%id, stvt%depth_flux%id, dimids(2:ndims)/),
00470                               varid=stvt%flux%varid)
00470          IF (nferr.NE.nf90_noerr) &
00471              CALL onf_error (caller='onf:invar', nffun='NF90_DEF_VAR', name=stvt%flux%name, nferr=nferr)
00472          nferr = nf90_def_var(ncid=stvid, name=stvt%soma%name, xtype=nf90_double,&
00473                               dimids=(/sms%id, dimids(1:ndims)/), varid=stvt%soma%varid)
00474          IF (nferr.NE.nf90_noerr) &
00475              CALL onf_error (caller='onf:invar', nffun='NF90_DEF_VAR', name=stvt%soma%name, nferr=nferr)
00476      END IF
00477      nferr = nf90_enddef(ncid=stvid)
00478      nferr = nf90_put_var(ncid=stvid, varid=stvt%depth%varid, values=stvt%depth%values)
00479      nferr = nf90_put_var(ncid=stvid, varid=stvt%depth%bounds%varid, values=stvt%depth%bounds%values)
00480      IF (ndims.EQ.4) THEN
00481          nferr = nf90_put_var(ncid=stvid, varid=stvt%lat%varid, values=stvt%lat%values)
00482          nferr = nf90_put_var(ncid=stvid, varid=stvt%lon%varid, values=stvt%lon%values)
00483      END IF
00484      IF (stvt%write_soma) THEN
00485          nferr = nf90_put_var(ncid=stvid, varid=sms%varid, values=fromto)

```

```

00486         DO n=1,stv%nsv_one_box
00487             nferr = nf90_put_var(ncid=stv%id, varid=stv%state%varid, values=trim(stv%var(n)%name), &
00488                 start=(/1, n/))
00489         END DO
00490         DEALLOCATE (ifrom, ito, ftnames, from, to, fromto)
00491         nferr = nf90_put_var(ncid=stv%id, varid=stv%depth_flux%varid, values=stv%flux_depth)
00492     END IF
00493 100 FORMAT ("Warning: cannot determine the location of the ",a," file relative to the output file. "/&
00494           "Either both or neither of the ",a," and output files should be absolute paths.")
00495 END SUBROUTINE invar
00496
00503 SUBROUTINE inpd (pdf, times, env, box, svd)
00504     use, INTRINSIC :: ieee_arithmetic
00505     IMPLICIT NONE
00506     CLASS(phydat), TARGET, INTENT(INOUT) :: pdf
00507     TYPE(timing), INTENT(INOUT) :: times
00508     TYPE(local), INTENT(INOUT) :: env
00509     TYPE(layer), INTENT(INOUT) :: box(0:)
00510     TYPE(nf90vec), DIMENSION(:), INTENT(IN) :: svd
00511     CHARACTER(LEN=NF90_MAX_NAME) :: descr
00512     INTEGER :: ind(1), iPAR=0, n, nn, nb, nsv, nferr
00513     INTEGER, DIMENSION(:), ALLOCATABLE :: ibot
00514     LOGICAL :: bottom
00515     IF ((ngdds.GT.0).AND.(pdf%ntim.EQ.1)) &
00516         stop 'inpd:specify temporally constant profiles as profile in namelist physics.'
00517     nsv = SIZE(svd)
00518     ALLOCATE (pdsds(pdf%ntim,nsfds+nsvds+nbvds), pdgds(pdf%ntim,ngdds), &
00519             pdf%attributes(nsfds+nsvds+nbvds+ngdds+npfds), pdf%sds(nbvds+nsvds+nsfds), &
00520             pdf%pds(pdf%ntim,ngdds), pdf%cds(pdf%ntim,npfds), pdf%isflx(nsfds), ibot(nbvds))
00521     DO nb=0,pdf%nbx
00522         ! nullify so that association status can be checked
00523         NULLIFY (box(nb)%temperature, box(nb)%alkalinity, box(nb)%salinity, box(nb)%vdc, &
00524                 box(nb)%velocity, box(nb)%silc)
00525     END DO
00526     pdf%surflux => pdf%sds(1:nsfds)
00527     pdf%botbc => pdf%sds(nsfds+1:nsfds+nbvds)
00528     ibot = 0d0
00529     DO n=1,ngdds
00530         ! gridded (depth-time) forcing data
00531         bottom = .false.
00532         ! whether data apply to the bottom of the box
00533         SELECT CASE (trim(pdf%data(n)%type))
00534             CASE ('Temperature')
00535                 pdf%data(n)%units = 'C'
00536                 DO nb=1,pdf%nbx; box(nb)%temperature => pdf%pds(nb,n); END DO
00537             CASE ('Salinity')
00538                 pdf%data(n)%units = 'psu'
00539                 DO nb=1,pdf%nbx; box(nb)%salinity => pdf%pds(nb,n); END DO
00540             CASE ('Alkalinity')
00541                 IF (env%ialk.GT.0) stop &
00542                     'Alkalinity is specified both in forcing in namelist physics and &
00543                     &in dima in namelist coco as a state variable. Please remove from&
00544                     & physics or coco.'
00545                 pdf%data(n)%units = 'mmol m-3'
00546                 DO nb=1,pdf%nbx; box(nb)%alkalinity => pdf%pds(nb,n); END DO
00547             CASE ('VDC')
00548                 ! vertical diffusivity at bottom of box
00549                 bottom = .true.
00550                 ! set to 0 at bottom of model domain if closed
00551                 pdf%data(n)%units = 'm2 s-1'
00552                 ALLOCATE (box(0)%vdc)
00553                 box(0)%vdc = 0d0
00554                 ! 0 at surface of model domain
00555                 DO nb=1,pdf%nbx; box(nb)%vdc => pdf%pds(nb,n); END DO
00556             CASE ('Velocity')
00557                 ! vertical velocity at bottom of box
00558                 bottom = .true.
00559                 ! set to 0 at bottom of model domain if closed
00560                 pdf%data(n)%units = 'm s-1'
00561                 ALLOCATE (box(0)%velocity)
00562                 box(0)%velocity = 0d0
00563                 ! 0 at surface of model domain
00564                 DO nb=1,pdf%nbx; box(nb)%velocity => pdf%pds(nb,n); END DO
00565             CASE DEFAULT
00566                 WRITE (stdout,110) 'data', trim(pdf%data(n)%type)
00567                 stop
00568         END SELECT
00569     WRITE (stdout,100) trim(pdf%data(n)%type), trim(pdf%data(n)%name)
00570     CALL pdf%data(n)%read (group=pdf, data=pdgds(:,n))
00571     CALL set_att (group=pdf, ds=pdf%data(n), natt=n)
00572     IF (pdf%clsbot.AND.bottom) pdgds(pdf%nbx,:,n) = 0d0
00573     END DO
00574     NULLIFY (env%sfDIC, env%sfO2)
00575     DO n=1,nsfds
00576         ! surface-flux data
00577         ind = pack([(nn, nn = 1,nsv)], mask=svd%name.EQ.trim(pdf%surflx(n)%type), vector=(/0/))
00578         IF (ind(1).GT.0) THEN
00579             descr = 'Surface flux of '//trim(pdf%surflx(n)%type)
00580             pdf%surflx(n)%units = trim(svd(ind(1))%units)//' m s-1'
00581             pdf%isflx(n) = ind(1) ! connect state-variable index (ind(1)) with surface data index (n)
00582             IF (ind(1).EQ.env%idic) env%sfDIC => pdf%surflux(n)
00583             IF (ind(1).EQ.env%io2) env%sfO2 => pdf%surflux(n)
00584         ELSE
00585             WRITE (stdout,110) 'surface', trim(pdf%surflx(n)%type)
00586             stop
00587         END IF
00588     WRITE (stdout,100) trim(descr), trim(pdf%surflx(n)%name)

```

```

00579     CALL pdf%surflx(n)%read (group=pdf, data=pdsds(:,n)) ! subroutine read_ds
00580     CALL set_att (group=pdf, ds=pdf%surflx(n), natt=n+ngdds)
00581   END DO
00582   IF (.NOT.ASSOCIATED(env%sfDIC)) ALLOCATE (env%sfDIC)
00583   IF (.NOT.ASSOCIATED(env%sfO2)) ALLOCATE (env%sfO2)
00584   env%sfDIC = 0._dp
00585   env%sfO2 = 0._dp
00586   DO n=1,nbvds ! bottom-boundary data
00587     ind = pack([(nn, nn = 1,nsv)], mask=svd$name.EQ.trim(pdf%bottom(n)%type), vector=(/0/))
00588     IF (ind(1).GT.0) THEN
00589       ibot(n) = ind(1)
00590       WRITE (stdout,100) 'bottom '//trim(pdf%bottom(n)%type), trim(pdf%bottom(n)%name)
00591       pdf%bottom(n)%units = svd(ind(1))%units
00592       CALL pdf%bottom(n)%read (group=pdf, data=pdsds(:,n+nsfds))
00593       CALL set_att (group=pdf, ds=pdf%bottom(n), natt=n+ngdds+nsfds)
00594     ELSE
00595       WRITE (stdout,110) 'bottom', trim(pdf%bottom(n)%type)
00596       stop
00597     END IF
00598   END DO
00599   ! names of surface datasets specified as surface types and names in
00600   ! namelist physics in subroutine innf (states%open in plankton:plankton_init)
00601   NULLIFY (pdf%surPAR)
00602   DO n=1,nsvds ! surface-value data
00603     nn = n + nbvds + nsfds
00604     SELECT CASE (trim(pdf%surface(n)%type))
00605     CASE ('avgPAR', 'avgSWR')
00606       descr = 'Average daily surface '//pdf%surface(n)%type(4:6)
00607       pdf%surface(n)%units = 'W m-2'
00608       DEALLOCATE (env%dayPAR)
00609       ALLOCATE (env%sPAR)
00610       env%dayPAR => pdf%sds(nn)
00611       env%calcPAR = .false.
00612       IF (ipar.NE.0) stop 'Please specify only one surface-irradiance dataset.'
00613       IF (trim(pdf%surface(n)%type).EQ.'avgSWR') ipar = nn
00614     CASE ('PAR', 'SWR')
00615       descr = 'Instantaneous surface '//pdf%surface(n)%type(1:3)
00616       pdf%surface(n)%units = 'W m-2'
00617       DEALLOCATE (env%dayPAR)
00618       env%sPAR => pdf%sds(nn)
00619       pdf%surPAR => pdsds(:,nn)
00620       env%calcPAR = .false.
00621       IF (ipar.NE.0) stop 'Please specify only one surface-irradiance dataset.'
00622       IF (trim(pdf%surface(n)%type).EQ.'SWR') ipar = nn
00623     CASE ('Wind')
00624       descr = 'Surface wind'
00625       pdf%surface(n)%units = 'm s-1'
00626       DEALLOCATE (env%wind)
00627       env%wind => pdf%sds(nn)
00628     CASE ('Ice')
00629       descr = 'Surface ice cover'
00630       pdf%surface(n)%units = '1'
00631       DEALLOCATE (env%ice)
00632       env%ice => pdf%sds(nn)
00633     CASE ('Pressure')
00634       descr = 'Surface pressure'
00635       pdf%surface(n)%units = 'Pa'
00636       DEALLOCATE (env%pressure)
00637       env%pressure => pdf%sds(nn)
00638     CASE ('CO2')
00639       descr = 'Atmospheric CO2'
00640       pdf%surface(n)%units = '1'
00641       DEALLOCATE (env%xco2)
00642       env%xco2 => pdf%sds(nn)
00643     CASE ('Alkalinity')
00644       IF (env%ialk.GT.0) stop &
00645         'Alkalinity is specified both in surface in namelist physics and in dima in &
00646         & namelist coco as a state variable. Please remove from physics or coco.'
00647       IF (ASSOCIATED(box(1)%alkalinity)) stop &
00648         'Alkalinity is specified both as forcing and surface in namelist &
00649         &physics. Please remove one of them.'
00650       descr = 'Surface alkalinity'
00651       pdf%surface(n)%units = 'mmol m-3'
00652       box(1)%alkalinity => pdf%sds(nn)
00653     CASE ('law')
00654       descr = 'Attenuance'
00655       pdf%surface(n)%units = 'm-1'
00656       env%lacw => pdf%sds(nn)
00657     CASE ('Latitude')
00658       descr = 'Geographic latitude'
00659       pdf%surface(n)%units = 'rad'
00660       times%latrad => pdf%sds(nn)
00661     CASE DEFAULT
00662       WRITE (stdout,110) 'surface', trim(pdf%surface(n)%type)
00663       stop
00664     END SELECT
00665     WRITE (stdout,100) trim(descr), trim(pdf%surface(n)%name)

```

```

00666     CALL pdf%surface(n)%read (group=pdf, data=pdsds(:,nn))
00667     CALL set_att (group=pdf, ds=pdf%surface(n), natt=nn+ngdds)
00668 END DO
00669 IF (ipar.GT.0) pdsds(:,ipar) = pdsds(:,ipar)*pdf%PAR ! convert SWR to PAR
00670 DO n=1,npfds ! temporally constant vertical profiles
00671     bottom = .false.
00672     SELECT CASE (trim(pdf%profile(n)%type))
00673     CASE ('Temperature')
00674         pdf%profile(n)%units = '°C'
00675         DO nb=1,pdf%nbbox; box(nb)%temperature => pdf%cds(nb,n); END DO
00676     CASE ('Salinity')
00677         pdf%profile(n)%units = 'psu'
00678         DO nb=1,pdf%nbbox; box(nb)%salinity => pdf%cds(nb,n); END DO
00679     CASE ('Alkalinity')
00680         pdf%profile(n)%units = 'mmol m-3'
00681         DO nb=1,pdf%nbbox; box(nb)%alkalinity => pdf%cds(nb,n); END DO
00682     CASE ('VDC')
00683         bottom = .true.
00684         pdf%profile(n)%units = 'm2 s-1'
00685         DO nb=1,pdf%nbbox; box(nb)%vdc => pdf%cds(nb,n); END DO
00686     CASE ('Velocity')
00687         bottom = .true.
00688         pdf%profile(n)%units = 'm s-1'
00689         ALLOCATE (box(0)%velocity)
00690         box(0)%velocity = 0d0
00691         DO nb=1,pdf%nbbox; box(nb)%velocity => pdf%cds(nb,n); END DO
00692     CASE DEFAULT
00693         WRITE (stdout,110) 'data', trim(pdf%profile(n)%type)
00694         stop
00695     END SELECT
00696     WRITE (stdout,100) 'constant ' //trim(pdf%profile(n)%type), trim(pdf%profile(n)%name)
00697     CALL pdf%profile(n)%read (group=pdf, data=pdf%cds(:,n))
00698     CALL set_att (group=pdf, ds=pdf%profile(n), natt=n+ngdds+nsfdds+nbvds)
00699     IF (pdf%clsbot.AND.bottom) pdf%cds(pdf%nbbox,n) = 0d0
00700 END DO
00701 IF (pdf%id.NE.0) THEN
00702     nferr = nf90_close(ncid=pdf%id)
00703     IF (pdf%ntim.GT.1) pdf%get => readpd
00704 END IF
00705 ALLOCATE (pdf%ibot(count(ibot.GT.0)))
00706 pdf%ibot = pack(ibot, mask=ibot.GT.0)
00707 IF (any(isnan(pdgds)).OR.any(isnan(pdsds))) THEN
00708     WRITE (stderr,('inpd: NaN detected in forcing data.'))
00709     stop
00710 END IF
00711 100 FORMAT ("inpd: ",a," dataset: ",a,".")
00712 110 FORMAT ("inpd: unknown ",a," type: ",a,".")
00713 CONTAINS
00714 SUBROUTINE set_att (group, ds, natt)
00715     IMPLICIT NONE
00716     CLASS(nf90group), INTENT(INOUT) :: group
00717     CLASS(nf90var), INTENT(IN) :: ds
00718     INTEGER, INTENT(IN) :: natt
00719     group%attributes(natt)%name = trim(ds%type) // ' dataset'
00720     group%attributes(natt)%value = trim(ds%name)
00721 END SUBROUTINE set_att
00722 SUBROUTINE inpd
00723
00726 SUBROUTINE read_ds (ds, group, data)
00727     IMPLICIT NONE
00728     CLASS(nf90group), INTENT(IN) :: group
00729     CLASS(nf90var), INTENT(INOUT) :: ds
00730     REAL(dp), DIMENSION(*), INTENT(OUT) :: data
00731     REAL(dp) :: cf, os
00732     CHARACTER(LEN=NF90_MAX_NAME) :: nf_units
00733     INTEGER :: nferr, nd, rank_ds, len, dids (NF90_MAX_VAR_DIMS)
00734     INTEGER, DIMENSION(:), ALLOCATABLE :: starts, counts
00735     nferr = nf90_inq_varid(ncid=group%id, name=ds%name, varid=ds%varid)
00736     IF (nferr.NE.nf90_noerr) &
00737         CALL onf_error (caller='read_ds ('//trim(group%filename)//')', &
00738             nffun='NF90_INQ_VARID', name=ds%name, nferr=nferr)
00739     nferr = nf90_inquire_variable(ncid=group%id, varid=ds%varid, ndims=rank_ds, dimids=dids)
00740     ALLOCATE (starts(rank_ds), counts(rank_ds))
00741     starts = 1
00742     counts = 0
00743     DO nd=1,rank_ds
00744         IF (dids(nd).EQ.group%lat%id) THEN
00745             starts(nd) = group%ilat
00746             counts(nd) = 1
00747         ELSEIF (dids(nd).EQ.group%lon%id) THEN
00748             starts(nd) = group%ilon
00749             counts(nd) = 1
00750         ELSEIF (dids(nd).EQ.group%depth%id) THEN
00751             counts(nd) = group%nbbox
00752         ELSEIF (dids(nd).EQ.group%bounds%id) THEN
00753             counts(nd) = 2
00754         ELSEIF (dids(nd).EQ.group%time%id) THEN

```

```

00755         starts(nd) = group%it0
00756         counts(nd) = group%ntim
00757     END IF
00758 END DO
00759 IF (count(counts.GT.0).NE.rank_ds) THEN
00760     WRITE (stderr, '("read_ds: rank of ",A," is ",I1," should be ",I1,".")') &
00761         trim(ds%name), rank_ds, count(counts.GT.0)
00762     stop
00763 END IF
00764 len = product(counts)
00765 nferr = nf90_get_var(ncid=group%id, varid=ds%varid, values=data(1:len), &
00766     start=starts, count=counts)
00767 IF (nferr.NE.nf90_noerr) &
00768     CALL onf_error (caller='read_ds', nffun='NF90_GET_VAR', name=ds%name, nferr=nferr)
00769 nferr = nf90_get_att(ncid=group%id, varid=ds%varid, name='units', values=nf_units)
00770 IF ((nferr.NE.nf90_noerr).AND.(nferr.NE.nf90_enotatt)) &
00771     CALL onf_error ("read_ds", "NF90_GET_ATT", trim(ds%name)//'/units', nferr)
00772 IF ((len_trim(ds%units).GT.0).AND.(nferr.NE.nf90_enotatt)) THEN
00773     CALL slot (nf_units, ds%units, slope=cf, offset=os)
00774     data(1:len) = data(1:len)*cf + os
00775 ELSEIF (len_trim(ds%units).GT.0) THEN
00776     WRITE (stdout, '("read_ds: Dataset ",A," in file ",A," has no units, assuming ",A,".")') &
00777         trim(ds%name), trim(group%filename), trim(ds%units)
00778 ELSEIF (nferr.EQ.nf90_enotatt) THEN
00779     WRITE (stdout, '("read_ds: Dataset ",A," in file ",A," has no units.")') &
00780         trim(ds%name), trim(group%filename)
00781     stop
00782 END IF
00783 END SUBROUTINE read_ds
00784
00786 SUBROUTINE readpd (pda, time)
00787     IMPLICIT NONE
00788     CLASS(phydat), INTENT(INOUT) :: pda
00789     REAL(dp), INTENT(IN) :: time
00790     INTEGER :: j
00791     INTEGER, SAVE :: ipdt=1
00792     REAL(dp) :: a, b
00793     DO WHILE ((time.LT.pda%time%values(ipdt)).AND.(ipdt.GT.1))
00794         ipdt = ipdt - 1
00795     END DO
00796     j = ipdt
00797     DO WHILE ((time.GT.pda%time%values(j)).AND.(j.LT.pda%ntim))
00798         j = j + 1
00799     END DO
00800     IF (ipdt.EQ.j) THEN
00801         pda%pds = pdgds(:,ipdt,:)
00802         pda%sds = pdsds(ipdt,:)
00803     ELSE
00804         b = (time - pda%time%values(ipdt))/(pda%time%values(j) - pda%time%values(ipdt))
00805         a = 1d0 - b
00806         pda%pds = a*pdgds(:,ipdt,:) + b*pdgds(:,j,:)
00807         pda%sds = a*pdsds(ipdt,:) + b*pdsds(j,:)
00808     END IF
00809     entry nopd(pda, time)
00810 END SUBROUTINE readpd
00811
00813 SUBROUTINE outsv (stv, times, env)
00814     IMPLICIT NONE
00815     CLASS(statevars), INTENT(INOUT) :: stv
00816     TYPE(timing), INTENT(IN) :: times
00817     TYPE(local), INTENT(IN) :: env
00818     INTEGER :: nferr
00819     stv%it = stv%it + 1
00820     nferr = nf90_put_var(ncid=stv%id, varid=stv%time%varid, &
00821         values=(/times%now - stv%timediff/), start=stv%it, count=(/1/))
00822     CALL stv%write_output (env, times)
00823 END SUBROUTINE outsv
00824
00825 SUBROUTINE write_output (stv, env, times)
00826     IMPLICIT NONE
00827     CLASS(statevars), INTENT(INOUT) :: stv
00828     TYPE(local), INTENT(IN) :: env
00829     TYPE(timing), INTENT(IN) :: times
00830     REAL(dp) :: dt
00831     INTEGER :: n, nferr
00832     dt = times%now - times%previous
00833     stv%all_states(stv%i0flux:stv%nsv_total) = stv%all_states(stv%i0flux:stv%nsv_total)/dt
00834     nferr = nf90_put_var(ncid=stv%id, varid=stv%flux%varid, values=stv%flux%values, &
00835         start=(/1, stv%start/), count=(/stv%nsv_one_box, stv%nfl, stv%count(2:)/))
00836     nferr = nf90_put_var(ncid=stv%id, varid=stv%soma%varid, values=stv%soma%values, &
00837         start=(/1, stv%start/), count=(/stv%nsms, stv%count/))
00838     stv%all_states(stv%nsv_all_boxes+1:stv%nsv_total) = 0._dp
00839     entry write_states(stv, env, times)
00840     DO n=1,env%nq0
00841         offsets(env%irt(env%i0r(n)),:) = stv%var(env%irc(env%i0r(n)))%values(1:stv%nbox)*env%Q0(n) !
00842         subsistence N, P, ...
00843     END DO

```

```

00843      DO n=1, stv%nsv_one_box
00844          nferr = nf90_put_var(ncid=stv%id, varid=stv%var(n)%varid, &
00845                          values=stv%var(n)%values(1:stv%nbox) + offsets(n,:), &
00846                          start=stv%start, count=stv%count)
00847      END DO
00848  END SUBROUTINE write_output
00849
00851  SUBROUTINE exof (stv)
00852      IMPLICIT NONE
00853      CLASS(statevars), INTENT(INOUT) :: stv
00854      INTEGER :: nferr
00855      DEALLOCATE (stv%var, stv%start, stv%count)
00856      nferr = nf90_close(ncid=stv%id)
00857  END SUBROUTINE exof
00858
00867  SUBROUTINE read_dim (dim, ncid, caller, nferr)
00868      IMPLICIT NONE
00869      CLASS(dimension), INTENT(INOUT) :: dim
00870      INTEGER, INTENT(IN) :: ncid
00871      CHARACTER(LEN=*), INTENT(IN) :: caller
00872      INTEGER, INTENT(OUT), OPTIONAL :: nferr
00873      INTEGER :: nferror, len, dimids(2)
00874      nferror = nf90_inq_varid(ncid, name=trim(dim%name), varid=dim%varid)
00875      IF (PRESENT(nferr)) THEN
00876          nferr = nferror
00877          IF (nferr.EQ.nf90_enotvar) RETURN ! variable not found
00878      END IF
00879      IF (nferror.NE.nf90_noerr) CALL onf_error ("read_dim from "//trim(caller), "NF90_INQ_VARID",
trim(dim%name), nferror)
00880      nferror = nf90_inquire_variable(ncid, varid=dim%varid, dimids=dimids(1:1))
00881      IF (nferror.NE.nf90_noerr) CALL onf_error ("read_dim from "//trim(caller),
"NF90_INQUIRE_VARIABLE", trim(dim%name), nferror)
00882      dim%id = dimids(1)
00883      nferror = nf90_inquire_dimension(ncid=ncid, dimid=dim%id, len=len)
00884      IF (nferror.NE.nf90_noerr) CALL onf_error ("read_dim from "//trim(caller),
"NF90_INQUIRE_DIMENSION", trim(dim%name), nferror)
00885      ALLOCATE (dim%values(len))
00886      nferror = nf90_get_var(ncid, varid=dim%varid, values=dim%values)
00887      IF (nferror.NE.nf90_noerr) CALL onf_error ("read_dim from "//trim(caller), "NF90_GET_VAR",
trim(dim%name), nferror)
00888      nferror = nf90_get_att(ncid, varid=dim%varid, name='units', values=dim%units)
00889      IF (nferror.EQ.nf90_enotatt) THEN
00890          WRITE (stdout, '("read_dim from ", A, ": no units attribute for ", A)') trim(caller),
trim(dim%name)
00891          dim%units = "
00892      ELSEIF (nferror.NE.nf90_noerr) THEN
00893          CALL onf_error ("read_dim from "//trim(caller), "NF90_GET_ATT",
trim(dim%name)//'.'//trim(dim%units), nferror)
00894      ENDIF
00895      nferror = nf90_get_att(ncid, varid=dim%varid, name='bounds', values=dim%bounds%name)
00896      IF (nferror.EQ.nf90_enotatt) THEN
00897          dim%bounds%name = "
00898      ELSE
00899          nferror = nf90_inq_varid(ncid, name=trim(dim%bounds%name), varid=dim%bounds%varid)
00900          IF (nferror.NE.nf90_noerr) &
00901              CALL onf_error ("read_dim from "//trim(caller), "NF90_INQ_VARID", trim(dim%bounds%name),
nferror)
00902          ALLOCATE (dim%bounds%values(2,len))
00903          nferror = nf90_get_var(ncid, varid=dim%bounds%varid, values=dim%bounds%values)
00904          IF (nferror.NE.nf90_noerr) &
00905              CALL onf_error ("read_dim from "//trim(caller), "NF90_GET_VAR", trim(dim%bounds%name),
nferror)
00906      END IF
00907  END SUBROUTINE read_dim
00908
00909  SUBROUTINE onf_error (caller, nffun, name, nferr)
00910      IMPLICIT NONE
00911      CHARACTER(LEN=*), INTENT(IN) :: caller, nffun, name
00912      INTEGER, INTENT(IN) :: nferr
00913      WRITE (stderr, ' (A, " : ", A, " (" , A, ") : ", A, ", nferr = ", I4)') caller, nffun, trim(name), &
trim(nf90_strerror(nferr)), nferr
00914      IF (nferr.EQ.-101) WRITE (stderr, ' ("File ", A, " is blocked by another program.")') trim(name)
00915      stop
00916  END SUBROUTINE onf_error
00917
00918  END MODULE onf

```

7.27 /Users/mpahlow/oppla/src/oppla.F90 File Reference

Functions/Subroutines

- integer function [catch_signal_2](#) (signal)
- program [oppla](#)

7.27.1 Function/Subroutine Documentation

7.27.1.1 `catch_signal_2()`

```
integer function oppla::catch_signal_2 (  
    integer signal )
```

Definition at line 133 of file [oppla.F90](#).

Referenced by [oppla\(\)](#).

Here is the caller graph for this function:



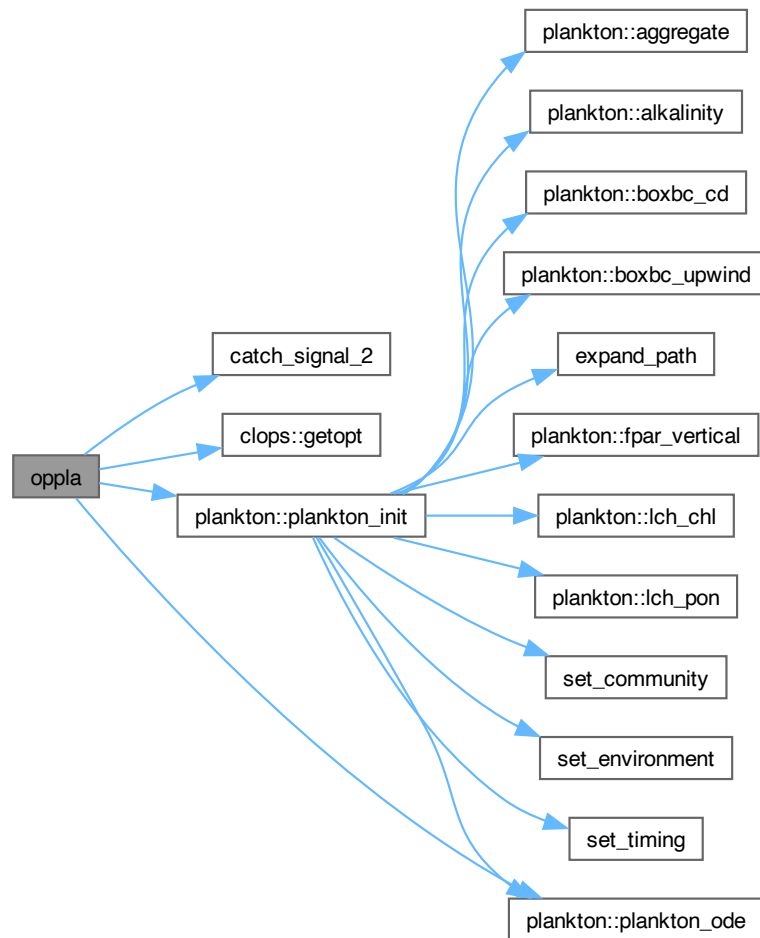
7.27.1.2 `oppla()`

```
program oppla
```

Definition at line 1 of file [oppla.F90](#).

References [catch_signal_2\(\)](#), [plankton::envir](#), [clops::getopt\(\)](#), [plankton::parode](#), [plankton::plankton_init\(\)](#), [plankton::plankton_ode\(\)](#), [plankton::states](#), and [plankton::times](#).

Here is the call graph for this function:



7.28 oppla.F90

[Go to the documentation of this file.](#)

```

00001 PROGRAM oppla
00002 #ifdef IFORT
00003   USE ifport
00004 #endif
00005   USE clops
00006   USE plankton
00007   USE dvode
00008   IMPLICIT NONE
00009   TYPE(vode_opts) :: options
00010   CHARACTER(LEN=*) , PARAMETER :: opts='diop:crfmv', &
00011     usage='oppla [-d data] [-i init] [-o outfile | -c | -r] [-p param] [-m] [-f] [-v]
00012     control_file', &
00013     help='(A/&
00014     &"-p param      (pfn) file with parameter namelists (default=control_file)"/&
00015     &"-d data       (dfn) file with physical data (netcdf)"/&
00016     &"-i init       (ifn) file with initial conditions (netcdf)"/&
00017     &"-o outfile    (ofn) output file (netcdf)"/&
00018     &"-c or -r     resume interrupted simulation"/&
00019     &"-f          overwrite output file if it exists"/&
00020     &"-m          (soma) write source and flux matrices to output file"/&
00021     &"-v          (quiet) show DVODE error messages"/&

```



```

00021      &"Names in parentheses correspond to namelist \"files\" in the input file,\"/&
00022      &"quiet is in namelist vode.\"/"Pressing CTRL-C stops integration.")'
00023      CHARACTER(LEN=NF90_MAX_NAME), TARGET :: fnms(5)="
00024      CHARACTER(LEN=NF90_MAX_NAME), POINTER :: dfn=>fnms(1), ifn=>fnms(2),&
00025      ofn=>fnms(3), pfn=>fnms(4), control=>fnms(5)
00026      INTEGER :: istate, ndil=1, nst=0, nts=0, nfe=0, exd, exh, exm,&
00027      exc0, exc1, excs, exct, ncps, istats(31)
00028      REAL(dp) :: exs, tout, tout_sun, tcrit, rstats(22)
00029      LOGICAL, TARGET :: flags(5)=.false.
00030      LOGICAL :: keep_going=.true., keep_going_main=.true.
00031      LOGICAL, POINTER :: resume=>flags(1), force=>flags(3), soma=>flags(4), verb=>flags(5)
00032      CALL system_clock (count=exc0, count_rate=ncps)
00033      excs = exc0
00034      #ifdef IFORT
00035      istate = signal(2, catch_signal_2, -1)
00036      #else
00037      istate = signal(2, catch_signal_2)
00038      #endif
00039      control = 'oppla_control.nml'
00040      CALL getopt (opts, fnms, flags, usage, help)
00041      parole%quiet = .NOT.verb
00042      resume = resume.OR.flags(2)
00043      CALL plankton_init (control=control, dfn0=dfn, ifn0=ifn, ofn0=ofn, pfn0=pfn, sms0=soma, force=force,
00044      resume=resume)
00045      times%timeout = times%timeout*ncps
00046      IF (parole%quiet) CALL xsetf (0)
00047      times%previous = times%now
00048      main: DO WHILE ((times%now.LT.times%end).AND.keep_going_main)
00049      ! integration should stop at sunrise, noon, and sunset, without creating
00050      ! excessive numbers of extra time steps
00051      ! first, check whether it is after sunset; if so, go to the next day
00052      IF ((times%now.GE.times%sun(4)).OR.(times%now.LT.(times%sun(4) - 86.4e3_dp))) &
00053      CALL times%sunday (env=envir)
00054      ! tout sun is the next sunrise, noon, or sunset after times%now
00055      tout_sun = minval(times%sun, mask=times%sun.GT.times%now)
00056      ! tout is the end of the next full time step or tout_sun
00057      tout = min(times%start + (nts + 1)*times%dto, tout_sun)
00058      ! if the next sunrise, noon, or sunset is less than times%dto after tout,
00059      ! extend tout until then
00060      IF ((tout_sun - tout).LT.times%dto) tout = tout_sun
00061      IF (times%dilute) tout = min(tout, times%dil_time(ndil))
00062      tcrit = tout
00063      options = set_opts(tcrit=tcrit, relerr_vector=parole%rtol, abserr_vector=parole%atol,&
00064      method_flag=parole%mf, h0=parole%h0, hmax=parole%hmax, hmin=parole%hmin,&
00065      maxord=parole%maxord, mxstep=parole%mxstep, mxhnil=parole%mxhnil,&
00066      constrained=parole%constrained, clower=parole%clower,&
00067      cupper=parole%cupper, sparse_j=parole%sparse_jacobian)
00068      DO WHILE ((times%now.LT.tout).AND.keep_going)
00069      istate = 1
00070      CALL vode_f90 (plankton_ode, parole%neq, states%all_states, times%now, tout, parole%itask,
00071      istate, options)
00072      SELECT CASE (istate)
00073      CASE (2)
00074      IF (options%mf.NE.parole%mf) &
00075      options = set_opts(tcrit=tcrit, relerr_vector=parole%rtol, abserr_vector=parole%atol,&
00076      method_flag=parole%mf, h0=parole%h0, hmax=parole%hmax, hmin=parole%hmin,&
00077      maxord=parole%maxord, mxstep=parole%mxstep, mxhnil=parole%mxhnil,&
00078      sparse_j=parole%sparse_jacobian, constrained=parole%constrained,&
00079      clower=parole%clower, cupper=parole%cupper)
00080      CALL system_clock (count=exc0)
00081      CASE (-1)
00082      ! excessive work: continue with diagonal Jacobian
00083      options = set_opts(tcrit=tcrit, relerr_vector=parole%rtol, abserr_vector=parole%atol,&
00084      method_flag=parole%jsv*(10*parole%meth + 3), h0=parole%h0, hmax=parole%hmax,&
00085      hmin=parole%hmin,&
00086      maxord=parole%maxord, mxstep=parole%mxstep, mxhnil=parole%mxhnil,&
00087      sparse_j=parole%sparse_jacobian,&
00088      constrained=parole%constrained, clower=parole%clower, cupper=parole%cupper)
00089      CASE (-5)
00090      ! convergence test failures
00091      ! temporarily switch to full Jacobian and lower accuracy
00092      options = set_opts(tcrit=tcrit, relerr_vector=parole%mxrtol, abserr_vector=parole%mxatol,&
00093      method_flag=parole%jsv*(10*parole%meth + 2), h0=parole%h0, hmax=parole%hmax,&
00094      hmin=parole%hmin,&
00095      maxord=parole%maxord, mxstep=parole%mxstep, mxhnil=parole%mxhnil,&
00096      sparse_j=parole%sparse_jacobian,&
00097      constrained=parole%constrained, clower=parole%clower, cupper=parole%cupper)
00098      CALL system_clock (count=exct)
00099      IF ((times%timeout.GT.0).AND.((exct - exc0).GT.times%timeout)) THEN
00100      WRITE (stdout, ' ("timeout reached")')
00101      EXIT main
00102      END IF
00103      CASE DEFAULT
00104      EXIT
00105      END SELECT
00106      IF (.NOT.parole%quiet) THEN
00107      CALL get_stats (rstats, istats)
00108      nst = nst + istats(11)
00109      nfe = nfe + istats(12)

```

```

00102         END IF
00103     END DO
00104     nts = floor((times%now + 1.e-3_dp - times%start)/times%dto)
00105     IF (nts.NE.nts0) THEN
00106         CALL states%write (times=times, env=envir) ! outsv in onf.f90
00107         times%previous = times%now
00108     END IF
00109     IF ((istate.NE.2).AND.(istate.NE.-1).AND.(istate.NE.-5)) THEN
00110         WRITE (*,/'("istate = ",I2," at time ",ES15.6)') istate, tout
00111         EXIT
00112     END IF
00113     IF (times%dilute) THEN
00114         IF (times%now.GE.times%dil_time(ndil)) THEN
00115             states%data = states%data*times%dil_fact(ndil)
00116             states%data(envir%idin,:) = states%data(envir%idin,:) + (1d0 -
times%dil_fact(ndil))*times%dil_DIN(ndil)
00117             states%data(envir%idip,:) = states%data(envir%idip,:) + (1d0 -
times%dil_fact(ndil))*times%dil_DIP(ndil)
00118             times%dilute = ndil.LT.times%ndil
00119             ndil = ndil + 1
00120         END IF
00121     END IF
00122     nts0 = nts
00123 END DO main
00124 CALL states%close ! exof in onf.f90
00125 IF (.NOT.parode%quiet) &
00126     WRITE (stdout,fmt=/'("No. of steps: ",I0,". No. of function calls: ",I0,".")') nst, nfe
00127 CALL system_clock(count=excl)
00128 CALL jul_dhms_ofsec((excl - exc0)/ncps, exd, exh, exm, excs)
00129 WRITE (stdout,/'("Integration took ",I0,":",I2.2,":",I2.2,".",I3.3)') exd*24 + exh, exm, excs, &
00130     int(mod(exs, 1._dp)*1.e3_dp)
00131 IF (istate.EQ.2) WRITE (stdout,fmt=/'("Oppla finished normally.")')
00132 CONTAINS
00133 FUNCTION catch_signal_2 (signal) RESULT (n)
00134     IMPLICIT NONE
00135     INTEGER :: signal, n
00136     keep_going = keep_going_main
00137     keep_going_main = .false.
00138     CALL xsetf (1)
00139     n = 0
00140 END FUNCTION catch_signal_2
00141 END PROGRAM oppla

```

7.29 /Users/mpahlow/oppla/src/plankton.f90 File Reference

Data Types

- type [plankton::bc](#)
Type for boundary conditions. [More...](#)

Modules

- module [plankton](#)
Set up and drive marine ecosystem dynamics.

Functions/Subroutines

- subroutine [plankton::aggregate](#) (env, grps, thisbox)
aggregation fluxes, forming detritus
- subroutine [plankton::alkalinity](#) (env, thisbox)
- subroutine [plankton::boxbc_cd](#) (env, boxes)
Boundary concentrations for modified central differences.
- subroutine [plankton::boxbc_upwind](#) (env, boxes)
boundary concentrations for upwind scheme
- character(len=nf90_max_name) function [expand_path](#) (file)

- subroutine `plankton::fpar_vertical` (env, boxes, nbox)
- subroutine `plankton::lch_chl` (ambient, boxes, stv)
light attenuation as a function of Chl
- subroutine `plankton::lch_pon` (ambient, boxes, stv)
light attenuation as a function of PON
- integer function `lfn` (fn)
length of filename without extension
- subroutine, public `plankton::plankton_init` (control, dfn0, ifn0, ofn0, pfn0, sms0, force, resume)
Read all parameters and boundary-condition information and initialise all modules.
- subroutine, public `plankton::plankton_ode` (neq, time, y, ydot)
Rates of change for all ODEs.
- subroutine `set_community` (lun, lunpar)
- subroutine `set_environment` (lun)
- subroutine `set_timing` (lun)

Variables

- type(`bacteria`), `dimension(:)`, allocatable, target `plankton::bacpla`
vector of bacteria groups
- type(`layer`), `dimension(:)`, allocatable `plankton::box`
- type(`detritus`), `dimension(:)`, allocatable, target `plankton::detrit`
vector of detritus groups
- type(`dicsys`), save `plankton::dics`
- type(`dom`), `dimension(:)`, allocatable, target `plankton::doma`
vector of DOM types
- type(`local`), target, save, public `plankton::envir`
- type(`state`), `dimension(:)`, allocatable `plankton::fungrp`
vector of functional groups
- type(`ode`), target, save, public `plankton::parode`
- type(`phycmo`), `dimension(:)`, allocatable, target `plankton::phypla`
vector of phytoplankton groups
- type(`phydat`), save `plankton::phys`
- type(`statevars`), target, save, public `plankton::states`
- type(`timing`), target, save, public `plankton::times`
- type(`zoocfo`), `dimension(:)`, allocatable, target `plankton::zoopla`
vector of zooplankton groups

7.29.1 Data Type Documentation

7.29.1.1 type `plankton::bc`

Type for boundary conditions.

Definition at line 17 of file `plankton.f90`.

Class Members

<code>character(len=100)</code>	name	variable name
<code>type(quantity)</code>	value	value of boundary condition

7.29.2 Function/Subroutine Documentation

7.29.2.1 `expand_path()`

```
character(len=nf90_max_name) function plankton_init::expand_path (
    character(len=nf90_max_name), intent(in) file ) [private]
```

Definition at line 733 of file [plankton.f90](#).

Referenced by [plankton::plankton_init\(\)](#).

Here is the caller graph for this function:



7.29.2.2 `lfn()`

```
integer function plankton_init::lfn (
    character(len=*), intent(in) fn ) [private]
```

length of filename without extension

Definition at line 497 of file [plankton.f90](#).

7.29.2.3 `set_community()`

```
subroutine plankton_init::set_community (
    integer, intent(in) lun,
    integer, intent(in) lunpar ) [private]
```

Definition at line 504 of file [plankton.f90](#).

References [plankton::bacpla](#), [plankton::detrit](#), [plankton::doma](#), [plankton::envir](#), [plankton::fungrp](#), [plankton::phypla](#), [plankton::states](#), and [plankton::zoopla](#).

Referenced by [plankton::plankton_init\(\)](#).

Here is the caller graph for this function:



7.29.2.4 set_environment()

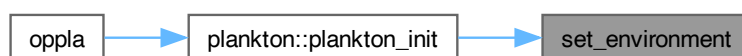
```
subroutine plankton_init::set_environment (
    integer, intent(in) lun ) [private]
```

Definition at line 634 of file [plankton.f90](#).

References [plankton::detrit](#), [plankton::dics](#), [plankton::envir](#), [plankton::states](#), and [plankton::times](#).

Referenced by [plankton::plankton_init\(\)](#).

Here is the caller graph for this function:



7.29.2.5 set_timing()

```
subroutine plankton_init::set_timing (
    integer, intent(in) lun ) [private]
```

Definition at line 705 of file [plankton.f90](#).

References [plankton::times](#).

Referenced by [plankton::plankton_init\(\)](#).

Here is the caller graph for this function:



7.30 plankton.f90

[Go to the documentation of this file.](#)

```

00001
00002 MODULE plankton
00003   USE et
00004   USE onf
00005   USE bac
00006   USE cmo
00007   USE det
00008   USE cfo
00009   USE dic
00010   USE julian
00011   USE brock81
00012   USE dnode
00013   IMPLICIT NONE
00014   PRIVATE
00015   PUBLIC :: plankton_init, plankton_ode, stdout, dp, nf90_max_name, jul_dhmsfsec
00017   TYPE :: bc
00018       CHARACTER(LEN=100) :: name
00019       TYPE(quantity) :: value
00020   END TYPE bc
00021   TYPE(ode), TARGET, SAVE, PUBLIC :: parode
00022   TYPE(timing), TARGET, SAVE, PUBLIC :: times
00023   TYPE(local), TARGET, SAVE, PUBLIC :: envvir
00024   TYPE(layer), ALLOCATABLE :: box(:)
00025   TYPE(state), ALLOCATABLE :: fungrp(:)
00026   TYPE(dom), ALLOCATABLE, TARGET :: doma(:)
00027   TYPE(bacteria), ALLOCATABLE, TARGET :: bacpla(:)
00028   TYPE(detritus), ALLOCATABLE, TARGET :: detrit(:)
00029   TYPE(phycmo), ALLOCATABLE, TARGET :: phypla(:)
00030   TYPE(zococo), ALLOCATABLE, TARGET :: zoopla(:)
00031   TYPE(dicsys), SAVE :: dics
00032   TYPE(phydat), SAVE :: phys
00033   TYPE(statevars), TARGET, SAVE, PUBLIC :: states
00034 CONTAINS
00035
00042 SUBROUTINE plankton_init (control, dfn0, ifn0, ofn0, pfn0, sms0, force, resume)
00043   IMPLICIT NONE
00044   CHARACTER(LEN=*) , INTENT(IN) :: control, dfn0, ifn0, ofn0, pfn0
00045   LOGICAL, INTENT(IN) :: force, & !< flag whether to overwrite output file
00046       sms0, & !< flag whether to write fluxes to output file
00047   resume
00048   TYPE(quantity), DIMENSION(15) :: time, din, dip
00049   TYPE(bc), ALLOCATABLE :: bottom(:), & ! bottom boundary conditions for open configurations
00050   fsfalk(:) ! alkalinity surface-flux factors
00051   CHARACTER(LEN=NF90_MAX_NAME) :: dfn="", ifn="", ofn="", pfn="", &
00052   utcdatstr, locdatstr, fmt, flux_name='flux', soma_name='soma', state_name='States'
00053   CHARACTER(LEN=512) :: iomsg
00054   LOGICAL :: soma=.false.
00055   LOGICAL, ALLOCATABLE :: ichl(:), ipic(:), mask(:)
00056   REAL(dp) :: factor(15)=1._dp, &
00057   flux_depth(100)=-1._dp
00058   REAL(dp), ALLOCATABLE :: faces(:), ydot(:) ! dummy
00059   INTEGER, ALLOCATABLE :: flux_level(:), position(:)
00060   INTEGER :: k, m, n, nagg, npic, nfalk, &
00061   nsfalk, & ! number of surface fluxes affecting alkalinity
00062   nbfish=0, & ! number of boxes (layers) with fish
00063   np(1), lunctl, lunp, pos, dutc, seconds, &
00064   nq0, nrs, nrft
00065   INTEGER(KIND=INT64) :: tsec
00066   namelist /files/ dfn, ifn, ofn, pfn
00067   namelist /dilute/ time, din, dip, factor
00068   namelist /fluxes/ soma, flux_depth, flux_name, soma_name, state_name
00069   CALL inudu ! initialise UDUNITS2 library (module fudu)
00070   OPEN (newunit=lunctl, file=expand_path(control), action='READ', status='OLD', iomsg=iomsg,
00071   err=900)
00072   pfn = control ! read everything from main input file by default
00073   READ (unit=lunctl, nml=files, END=10, IOMSG=iomsg, ERR=900)
00074   10 CONTINUE
00075   IF (len_trim(pfn).GT.0) pfn = pfn0 ! command line overrides namelist files
00076   IF (len_trim(dfn).GT.0) dfn = dfn0
00077   IF (len_trim(ifn).GT.0) ifn = ifn0
00078   IF (len_trim(ofn).GT.0) ofn = ofn0
00079   IF (len_trim(ifn).EQ.0) stop 'Please specify initial-condition file (option -i or ifn in namelist
00080   files).'
00081   IF (len_trim(ofn).EQ.0) stop 'Please specify output file (option -o or ofn in namelist files).'
00082   IF (resume) ifn = ofn
00083   CALL states%open (phys=phys, lun=lunctl) ! initialise NetCDF library (innf in module onf)
00084   ! read community composition from namelist coco
00085   IF (trim(pfn).EQ.trim(control)) THEN
00086       lunp = lunctl
00087   ELSE
00088       OPEN (newunit=lunp, file=expand_path(pfn), action='READ', status='OLD', iomsg=iomsg, err=900)
00089   END IF

```

```

00088      rewind(unit=lunp)
00089      envirncon = SIZE(constituents) ! number of constituents (C, N, P, Chl)
00090      CALL set_community (lun=lunctl, lunpar=lunp)
00091      CALL set_environment (lun=lunctl)
00092      CALL set_timing (lun=lunctl)
00093      states%filename = expand_path(ifn) ! file with initial conditions
00094      states%timediff = times%dtutc
00095      ! states%info => nf90info in module onf (TYPE(nf90group) -> TYPE(statevars))
00096      CALL states%info (latdeg=times%latdeg, londeg=times%londeg, t0=times%tin, writeable=resume)
00097      IF (resume) times%start = states%time%values(1)
00098      IF (len_trim(dfn).EQ.0) THEN ! no physical forcing data
00099          ALLOCATE (phys%depth%values(states%nbox))
00100          phys%depth = states%depth ! apply depth structure of
00101          phys%nbox = states%nbox ! initial conditions
00102      ELSE
00103          phys%filename = expand_path(dfn) ! name of forcing file
00104          phys%timediff = times%dtutc
00105          ! phys%info => nf90info (TYPE(nf90group) -> TYPE(phydat) in module onf)
00106          CALL phys%info (latdeg=times%latdeg, londeg=times%londeg, t0=times%start, t1=times%end)
00107          states%depth%bounds%values = phys%depth%bounds%values
00108      ENDIF
00109      envirnbox = phys%nbox
00110      envir%depth_edges => phys%depth%bounds%values
00111      rewind(unit=lunctl)
00112      READ (unit=lunctl, nml=fluxes, END=20, IOMSG=iomsg, ERR=900)
00113 20 CONTINUE
00114      states%write_soma = soma.OR.sms0
00115      IF (states%write_soma) THEN
00116          WRITE (*, '("plankton_init: writing fluxes to output file.")')
00117          ALLOCATE (faces(phys%nbox+1), envir%fmsk(1:states%nsv_one_box+1:phys%nbox+1))
00118          states%state%name = trim(state_name)
00119          states%flux%name = trim(flux_name)
00120          states%soma%name = trim(soma_name)
00121          states%nfl = count(flux_depth.GE.0._dp)
00122          faces = (/envir%depth_edges(1,:), envir%depth_edges(2,phys%nbox)/)
00123          IF (states%nfl.EQ.0) THEN
00124              states%nfl = phys%nbox + 1
00125              flux_depth(1:states%nfl) = faces
00126              envir%fmsk = .true. ! flux-matrix mask
00127          ELSE
00128              envir%fmsk = .false. ! flux-matrix mask
00129          END IF
00130          ALLOCATE (flux_level(states%nfl), states%flux_depth(states%nfl))
00131          DO n=1,states%nfl
00132              flux_level(n) = minloc(abs(faces - flux_depth(n)), dim=1, back=.true.)
00133              envir%fmsk(:,flux_level(n)) = .true.
00134          END DO
00135          states%flux_depth = faces(flux_level)
00136          DEALLOCATE (faces, flux_level)
00137      END IF
00138      ! rows 0 of soma are phantom rows for aggregation fluxes
00139      ! traits are stored and transported as tracer-ratio products
00140      ALLOCATE (envir%quotas(envirncon,envirnft), envir%qmsk(envirncon,envirnft), &
00141          envir%Q0(envirnq0), envir%iq0r(envirnq0), &
00142          envir%ratios(envirnrs), envir%all_ratios(envirnrs,phys%nbox), &
00143          envir%OC(envirnft), envir%soma(0:states%nsv_one_box,states%nsv_one_box,phys%nbox), &
00144          envir%smask(0:states%nsv_one_box,states%nsv_one_box,phys%nbox), &
00145          envir%irc(envirnrs), envir%irt(envirnrs), envir%vvs(states%nsv_one_box), &
00146          envir%vvstv(states%nsv_one_box), envir%dydz(states%nsv_one_box), envir%iagg(envirnagg), &
00147          envir%sticky(envirnagg), envir%kij(envirnagg,envirnagg), envir%agg(envirnagg), &
00148          envir%heights(states%nsv_one_box, phys%nbox), envir%depth(0:phys%nbox+1), &
00149          box(0:phys%nbox+1), ichl(states%nsv_one_box), ipic(states%nsv_one_box), &
00150          envir%PAR(0:phys%nbox), envir%fPAR(0:phys%nbox))
00151      envir%depth = (/0._dp, phys%depth%values, 0._dp/)
00152      box%depth = envir%depth
00153      box(0)%height = 0._dp
00154      box%nb = (/ (n, n=0,phys%nbox+1)/) ! layer index
00155      envir%PARfrac => fpar_vertical
00156      DO n=1,phys%nbox ! convert mean depths to box heights
00157          box(n)%dz = box(n)%depth - box(n-1)%depth
00158          box(n)%height = (box(n)%dz - box(n-1)%height)
00159      END DO
00160      box(1:phys%nbox)%height = 2d0*box(1:phys%nbox)%height
00161      envir%heights = spread(box(1:phys%nbox)%height, dim=1, ncopies=states%nsv_one_box) ! layer heights
00162      ! rdl: ratio of lower box height to total height of two boxes
00163      box(0:phys%nbox-1)%rdl = 1d0/(box(0:phys%nbox-1)%height/box(1:phys%nbox)%height + 1d0)
00164      box(phys%nbox)%rdl = 0.5d0
00165      box(phys%nbox+1)%nb = phys%nbox + 1
00166      box(1:nbfish)%fish = 1
00167      envir%vvs = 0d0 ! vertical velocities (sinking, DVM, SVM)
00168      envir%quotas = 0d0 ! default
00169      envir%quotas(1,:) = 1d0 ! the C quota (C:C) is always 1
00170      envir%qmsk = .false. ! quota mask
00171      envir%smask = .false. ! source-matrix mask
00172      envir%soma = 0d0 ! source matrices (fluxes between states)
00173      nq0 = 0
00174      nrs = 0

```

```

00175     nagg = 0
00176     DO n = 1, envnr%nfnt
00177         ! number of functional types
00177         fungrp(n)%var%vv => envnr%vvs(fungrp(n)%var%isv(1):fungrp(n)%var%isv(fungrp(n)%var%nsv))
00178         fungrp(n)%var%vv = 0d0 ! vertical velocities for each state in each type
00179         fungrp(n)%var%OC => envnr%OC(n) ! organic carbon (POC)
00180         fungrp(n)%var%QN => envnr%quotas(2,n) ! N:C quota
00181         fungrp(n)%var%QP => envnr%quotas(3,n) ! P:C quota
00182         NULLIFY (fungrp(n)%var%stick)
00183         IF (fungrp(n)%var%sticky.GT.0d0) THEN ! does group n undergo aggregation?
00184             nagg = nagg + 1
00185             envnr%iagg(nagg) = n ! indices of aggregating types
00186             fungrp(n)%var%stick => envnr%sticky(nagg) ! stickiness
00187             fungrp(n)%var%stick = fungrp(n)%var%sticky
00188         END IF
00189         IF (fungrp(n)%var%nsv.GT.1) THEN ! dynamic quotas or tracer-trait products
00190             nrft = fungrp(n)%var%nsv - 1
00191             envnr%irc(nrs+1:nrs+nrft) = fungrp(n)%var%isv(1) ! ratio reference (POC) tracer-indices
00192             envnr%irt(nrs+1:nrs+nrft) = fungrp(n)%var%isv(2:) ! ratio tracer-indices
00193             fungrp(n)%var%ratios => envnr%ratios(nrs+1:nrs+nrft)
00194             fungrp(n)%var%ir = (/ (m + nrs, m=1,nrft) /)
00195             IF (fungrp(n)%var%nq0.GT.0) THEN ! subsistence quotas
00196                 envnr%iqr(nq0+1:nq0+fungrp(n)%var%nq0) = nrs + fungrp(n)%var%iqr - 1
00197                 envnr%Q0(nq0+1:nq0+fungrp(n)%var%nq0) = fungrp(n)%var%Q0 ! assign subsistence quota
00198                 nq0 = nq0 + fungrp(n)%var%nq0
00199             END IF
00200             DO m=2, fungrp(n)%var%ncon ! associate dynamic quotas with constituents
00201                 np = pack([(k, k=2, envnr%ncon)], &
00202                     mask=trim(fungrp(n)%var%constituents(m)).EQ.constituents(2:), vector=(/0/))
00203                 IF (np(1).EQ.0) THEN
00204                     WRITE (stderr, ' ("Constituent ", A, " not implemented!") ' ) &
00205                         trim(fungrp(n)%var%constituents(m))
00206                     stop
00207                 END IF
00208                 envnr%qmsk(np(1),n) = .true. ! quota mask (np(1) = constituent index, n = group index)
00209             END DO
00210             nrs = nrs + nrft
00211         END IF
00212         ! fungrp(n)%var%set points to the set procedure pointer in each functional group
00213         ! for example, if fungrp(n)%var is of TYPE(phycmo) (defined in module cmo), then
00214         ! fungrp(n)%var%set => phypla%set => cmo_set, defined in module cmo
00215         CALL fungrp(n)%var%set (fungrp, envnr) ! initialise group n
00216         ! names and units of all states in functional group n
00217         states%var(fungrp(n)%var%isv)%name = fungrp(n)%var%names
00218         states%var(fungrp(n)%var%isv)%units = fungrp(n)%var%units
00219     END DO
00220     ! set bottom boundary conditions from bottom in namelist envi
00221     ALLOCATE(mask(states%nsv_one_box))
00222     DO n=1, count(bottom%name.NE."")
00223         mask = states%var%name.EQ.bottom(n)%name
00224         IF (any(mask)) THEN
00225             np = pack(position, mask=mask)
00226             envnr%bbc(np(1)) = convert(bottom(n)%value, states%var(np(1))%units)
00227         ELSE
00228             WRITE (stderr, ' ("Ignoring bottom value for non-existent variable ", A, ".") ' ) &
00229                 trim(bottom(n)%name)
00230         END IF
00231     END DO
00232     IF ((detrit(envnr%idagg)%kag.GT.0d0).AND.(envnr%nagg.GT.0)) THEN ! does aggregation occur?
00233         envnr%aggregate => aggregate
00234         envnr%kij = detrit(envnr%idagg)%kag
00235         DO n = 1, envnr%nagg
00236             envnr%kij(n,n) = 2._dp*envnr%kij(n,n)
00237             envnr%smask(0, fungrp(envnr%iagg(n))%var%isv,:) = .true.
00238         END DO
00239         envnr%smask(0, detrit(1)%isv,:) = .true.
00240     ELSE
00241         envnr%aggregate => no_aggregate
00242     END IF
00243     states%smask => envnr%smask(:, :, 1)
00244     states%nsms = count(states%smask)
00245     states%nsv_all_boxes = states%nsv_one_box*phys%inbox
00246     IF (states%write_soma) THEN
00247         states%nsv_total = states%nsv_all_boxes + states%nsv_one_box*states%nfl + states%nsms*phys%inbox
00248         states%i0flux = states%nsv_all_boxes + 1
00249         states%i0sms = states%i0flux + states%nsv_one_box*states%nfl
00250     ELSE
00251         states%nsv_total = states%nsv_all_boxes
00252     END IF
00253     parode%neq = states%nsv_total
00254     CALL parode%read (lun=lunctl, nconstr=states%nsv_all_boxes) ! => read_ode in module dvoid
00255     IF (lunp.NE.lunctl) CLOSE (unit=lunctl)
00256     IF (trim(states%time%units).NE.trim(phys%time%units)) THEN
00257         WRITE (stdout, ' ("Time units in ", A, " are ", A, " and ", A, " in ", A, "!") ' ) &
00258             trim(phys%filename), trim(phys%time%units), trim(states%time%units), trim(states%filename)
00259         IF (resume) stop 'This should not happen for a continuation!'
00260     END IF
00261     IF (times%end.LE.times%start) stop 'plankton_init: end must be greater than start in namelist

```



```

timing.'
00262     states%time%units = phys%time%units
00263     IF (states%ilat.GT.0) THEN
00264         states%lat%values(states%ilat) = times%latdeg ! set lat/lon for output file
00265         states%lon%values(states%ilon) = times%londeg
00266     END IF
00267     ! states%init => invar (module onf): create or open output file; read initial conditions
00268     ! states%nsms and states%smk must be completely defined at this point
00269     CALL states%init (ofn=expand_path(ofn), control=expand_path(control), &
00270         dfn=expand_path(dfn), pfn=expand_path(pfn), force=force, resume=resume)
00271     CALL parode%set_atol(stv=states%data) ! => set_atol (module dvoid)
00272     NULLIFY (times%latrad)
00273     CALL phys%init (env=envir, times=times, box=box, svd=states%var) ! => inpd (module onf)
00274     IF (.NOT.ASSOCIATED(phys%surPAR)) CALL brock81_init (envir=envir, times=times)
00275     IF (phys%clsbot) box(phys%nbox)%botperm = 0d0
00276     ALLOCATE (envir%fluxes(1:states%nsv_one_box,0:phys%nbox))
00277     envir%fluxes = 0d0 ! fluxes between boxes (layers)
00278     rewind(unit=lunp)
00279     ! dics%read => indic (module dic): initialise DIC system
00280     CALL dics%read (lun=lunp, env=envir, fluxes=envir%fluxes)
00281     ! check for presence of datasets for alkalinity, temperature, etc.
00282     ! otherwise use constants supplied via namelist envi
00283     IF (.NOT.ASSOCIATED(box(1)%alkalinity)) THEN
00284         IF (envir%ialk.GT.0) THEN
00285             DO n=1,phys%nbox; box(n)%alkalinity => states%data(envir%ialk,n)
00286             END DO
00287         ELSEIF (dics%totalk.GT.0d0) THEN
00288             DO n=1,phys%nbox; ALLOCATE (box(n)%alkalinity); box(n)%alkalinity = dics%totalk
00289             END DO
00290         ELSEIF (dics%potalk.LE.0d0) THEN
00291             stop 'plankton_init: no alkalinity data found.'
00292         ELSE
00293             DO n=1,phys%nbox; ALLOCATE (box(n)%alkalinity)
00294             END DO
00295         END IF
00296     END IF
00297     IF (.NOT.ASSOCIATED(box(1)%temperature)) THEN
00298         DO n=1,phys%nbox; box(n)%temperature => envir%temperature
00299         END DO
00300     END IF
00301     IF (.NOT.ASSOCIATED(box(1)%salinity)) THEN
00302         DO n=1,phys%nbox; box(n)%salinity => envir%salinity
00303         END DO
00304     END IF
00305     IF (.NOT.ASSOCIATED(box(1)%vdc)) THEN
00306         ALLOCATE (box(0)%vdc); box(0)%vdc = 0d0
00307         DO n=1,phys%nbox; box(n)%vdc => envir%vdc
00308         END DO
00309     END IF
00310     IF (.NOT.ASSOCIATED(box(1)%velocity)) THEN
00311         ALLOCATE (box(0)%velocity); box(0)%velocity = 0d0
00312         DO n=1,phys%nbox; box(n)%velocity => envir%velocity
00313         END DO
00314     END IF
00315     envir%fPAR(0) = 1d0 ! fraction of surface irradiance (PAR) arriving at each layer
00316     SELECT CASE (trim(phys%advect)) ! select advection scheme
00317     CASE ('CD', 'cd') ! central differences (modified to be positive-definite)
00318         envir%boxbc => boxbc_cd
00319     CASE ('upwind')
00320         envir%boxbc => boxbc_upwind
00321     CASE ('dummy') ! no advection (only mixing)
00322         envir%boxbc => boxbc_dummy
00323     CASE DEFAULT
00324         stop 'plankton_init: invalid advection scheme (advect in namelist physics).'
00325     END SELECT
00326     IF (len_trim(phys%time%units).EQ.0) THEN ! no forcing
00327         dutc = 0
00328         seconds = 0
00329     ELSE
00330         pos = scan(trim(phys%time%units), '123456789')
00331         ! time%units should be something like "seconds since 1990-01-01"
00332         CALL jul_parsedt(trim(phys%time%units(pos:)), '%Y-%m-%d', dutc, seconds)
00333         ! dutc is the UTC day of the date in time%units
00334     END IF
00335     times%tbase = dutc*cfds + seconds ! date (in s) of the date in phys%time%units
00336     ! envir%PARfun is one of the PAR functions selected according to envir%dcy above
00337     CALL envir%PARfun (times) ! initialise light
00338     ! time and date in UTC
00339     tsec = nint(times%start + times%tbase - times%dtutc, kind=int64)
00340     dutc = floor(real(tsec, kind=dp)/cfds)
00341     seconds = int(modulo(tsec, cfds), kind=4)
00342     utcdatstr = jul_formatdate(dutc, seconds, fmt='%d %m %Y at %H:%M:%S')
00343     ! local time and date
00344     tsec = nint(times%start + times%tbase, kind=int64)
00345     dutc = floor(real(tsec, kind=dp)/cfds)
00346     seconds = int(modulo(tsec, cfds), kind=4)
00347     locdatstr = jul_formatdate(dutc, seconds, fmt='%d %m %Y at %H:%M:%S')

```

```

00348     WRITE (stdout, ' ("plankton_init: Starting simulation on ", A, " (" , A, " UTC).")') &
00349         trim(locdatstr), trim(utcdatstr)
00350     n = min(states%nbox, phys%nbox)
00351     IF ((states%nbox.NE.phys%nbox).OR.(any(abs(states%depth%values(1:n) -
envir%depth(1:n)).GT.1.e-12_dp))) &
00352         stop 'plankton_init: Depth structures in (re)start and data files must agree.'
00353     box(0)%flux => envir%fluxes(:,0) ! local fluxes (between boxes)
00354     DO n=1,phys%nbox
00355         box(n)%stv => states%data(:,n) ! all states in layer
00356         box(n)%tPAR => envir%PAR(n-1) ! PAR at top of layer
00357         box(n)%bPAR => envir%PAR(n) ! PAR at bottom of layer
00358         box(n)%DIN => states%data(envir%idin,n)
00359         box(n)%DIP => states%data(envir%idip,n)
00360         box(n)%DIC => states%data(envir%idic,n)
00361         box(n)%O2 => states%data(envir%io2,n)
00362         IF (envir%irdoc.GT.0) box(n)%rDOC => states%data(envir%irdoc,n) ! refractory DOC
00363         IF (envir%irdon.GT.0) box(n)%rDON => states%data(envir%irdon,n)
00364         box(n)%flux => envir%fluxes(:,n) ! local fluxes (between boxes)
00365         box(n)%soma(0:states%nsv_one_box,1:states%nsv_one_box) => envir%soma(:,n) ! local source
matrix (in 1 box)
00366         box(n)%ratios => envir%all_ratios(:,n)
00367         ALLOCATE (box(n)%bbc(states%nsv_one_box))
00368         box(n)%bbc = 0._dp
00369     END DO
00370     IF (envir%isi.NE.0) THEN
00371         DO n=1,phys%nbox; box(n)%silc => states%data(envir%isi,n)
00372         END DO
00373     ELSEIF (.NOT.ASSOCIATED(box(1)%silc)) THEN
00374         DO n=1,phys%nbox; ALLOCATE (box(n)%silc); box(n)%silc => box(n)%DIN
00375         END DO
00376     END IF
00377     ALLOCATE (box(0)%bbc(states%nsv_one_box), box(phys%nbox+1)%bbc(states%nsv_one_box), &
00378         box(0)%stv(states%nsv_one_box), box(phys%nbox+1)%stv(states%nsv_one_box), &
00379         box(0)%ratios(nrs), box(phys%nbox+1)%ratios(nrs))
00380     box(0)%stv = 0._dp ! ensure that surface concentrations are defined
00381     box(0)%bbc = 0._dp
00382     box(phys%nbox)%bbc = envir%bbc ! set bottom concentrations
00383     box(phys%nbox+1)%stv = envir%bbc
00384     box(0)%ratios = 0._dp
00385     box(phys%nbox+1)%ratios = 0._dp
00386     ichl = index(states%var%name, '_Chl').GT.0
00387     IF (any(ichl)) THEN
00388         ALLOCATE (envir%ichl(count(ichl)))
00389         envir%ichl = pack([(k, k=1,states%nsv_one_box)], mask=ichl) ! indices of all Chl states
00390         envir%attenuation => lch_chl
00391         IF (phys%nbox.EQ.1) THEN
00392             DO n=1,envir%nphy ! assign initial Chl if specified
00393                 IF (phypla(n)%Chl0.GT.0._dp) states%data(envir%ichl(n),1) = phypla(n)%Chl0
00394             END DO
00395         END IF
00396     ELSE
00397         DO n=1,envir%nphy
00398             envir%ipty(n) = fungrp(envir%ipty(n))%var%isv(2)
00399         END DO
00400         envir%attenuation => lch_pon
00401     END IF
00402     IF (envir%ialk.GT.0) THEN ! alkalinity is a state variable
00403         ipic = index(states%var%name, '_PIC').GT.0
00404         npic = count(ipic)
00405         nfalk = 3 + npic
00406         ALLOCATE (envir%ipic(npic), envir%ifalk(nfalk), envir%fALK(nfalk))
00407         envir%ifalk(1:3) = (/envir%idic, envir%idip/)
00408         ! account for S uptake assuming an S:C ratio of 0.0226 (Wolf-Gladrow et al. 2007,
00409         ! citing Matrai and Keller 1994), which may be less variable than the S:P ratio
00410         envir%fALK(1:3) = (/ -0.045_dp, -1._dp, -1._dp/)
00411         IF (any(ipic)) THEN
00412             envir%ipic = pack([(k, k=1,states%nsv_one_box)], mask=ipic) ! indices of all PIC states
00413             envir%ifALK(4:3+npic) = envir%ipic ! the 0.045 compensates for the spurious S uptake
00414             envir%fALK(4:3+npic) = -2.045_dp ! incurred by fALK(1) for the associated DIC change
00415         END IF
00416         ! surface fluxes
00417         nsfalk = count(fsfalk%name.NE."")
00418         IF ((.NOT.any(fsfalk%name.EQ.'alkalinity')).AND.&
00419             any(states%var(phys%isflx)%name.EQ.'alkalinity')) THEN
00420             nsfalk = nsfalk + 1
00421             fsfalk(nsfalk) = bc('alkalinity', quantity(1._dp, '1'))
00422         END IF
00423         n = 1
00424         DO WHILE (n.LE.nsfalk)
00425             IF (.NOT.any(fsfalk(n)%name.EQ.states%var(phys%isflx)%name)) THEN
00426                 WRITE (stderr, ' ("Ignoring alkalinity surface-flux factor for non-existent flux ", A, ".")')
&
00427                 trim(fsfalk(n)%name)
00428                 fsfalk(n) = fsfalk(nsfalk)
00429                 nsfalk = nsfalk - 1
00430             ELSE
00431                 n = n + 1

```

```

00432         END IF
00433     END DO
00434     envir%xsfsalk = ((fsfalk(1)%name.NE.'alkalinity').AND.(nsfalk.EQ.1)) &
00435         .OR.(nsfalk.GT.1)
00436     IF (envir%xsfsalk) THEN
00437         ALLOCATE (envir%isfalk(nsfalk), envir%fsfALK(nsfalk))
00438         DO n=1,nsfalk
00439             envir%isfALK(n:n) = pack(phys%isflx, mask=states%var(phys%isflx)%name.EQ.fsfalk(n)%name)
00440             envir%fsfALK(n) = convert(fsfalk(n)%value, '1')
00441         END DO
00442     END IF
00443     envir%alkalinity => alkalinity
00444 ELSE
00445     envir%alkalinity => no_alkalinity
00446 END IF
00447 time = quantity(0d0, 'd')
00448 din = quantity(0d0, 'mmolN m-3')
00449 dip = quantity(0d0, 'mmolP m-3')
00450 rewind(unit=lunp)
00451 ! this is for the PU1, 2 experiments, where the mesocosms were diluted
00452 READ (unit=lunp, nml=dilute, END=30, IOMSG=iomsg, ERR=900)
00453 times%ndil = count(time%value.GT.0d0) ! number of dilution events
00454 times%dilute = times%ndil.GT.0 ! is dilution done?
00455 ALLOCATE (times%dil_time(times%ndil), times%dil_fact(times%ndil), &
00456     times%dil_DIN(times%ndil), times%dil_DIP(times%ndil))
00457 DO n=1,times%ndil
00458     times%dil_time(n) = convert(time(n), 's') + times%start ! times of dilutions
00459     times%dil_fact(n) = factor(n) ! dilution factors
00460     times%dil_DIN(n) = convert(din(n), 'mmolN m-3') ! concentration in added water
00461     times%dil_DIP(n) = convert(dip(n), 'mmolP m-3')
00462 END DO
00463 30 CONTINUE
00464 CLOSE (unit=lunp)
00465 DO n=1,phys%nbbox ! check consistency of initial conditions
00466     IF (any(states%data(envir%irt(envir%iqr),n).LT.states%data(envir%irc(envir%iqr),n)*envir%Q0))
00467 THEN
00468         WRITE (fmt, "("&"Initial quota(s) too low in layer ",I0," for ",I0," (A:"&"", ""))") &
00469             SIZE(envir%iqr)
00470         WRITE (stderr,trim(fmt)) n, pack((/trim(states%var(envir%irt(envir%iqr(m))%name), m=1,&
00471             SIZE(envir%iqr)/)),
00472             mask=states%data(envir%irt(envir%iqr),n).LT.states%data(envir%irc(envir%iqr),n)*envir%Q0)
00471         stop
00472     END IF
00473     ! delta-tracers for PON, POP, ... with subsistence quotas to have a lower
00474     ! limit of 0 -> subtract subsistence N, P, ...
00475     states%data(envir%irt(envir%iqr),n) = states%data(envir%irt(envir%iqr),n) &
00476         - states%data(envir%irc(envir%iqr),n)*envir%Q0
00477 END DO
00478 ! write initial conditions (and fluxes) to output file
00479 times%now = times%start
00480 CALL times%sunday (env=envir)
00481 CALL phys%get (time=times%start)
00482 CALL dics%init (box=box) ! set initial [H+]
00483 IF (states%write_soma) states%all_states(states%i0flux:states%nsv_total) = 0._dp
00484 ALLOCATE (ydot(states%nsv_total))
00485 CALL plankton_ode (neq=states%nsv_total, time=times%start, y=states%all_states, ydot=ydot)
00486 ! write initial conditions to output file unless it is a continuation
00487 times%now = times%start
00488 times%previous = times%now - times%dto ! dummy time step for initial fluxes and sms (these are
all 0 here)
00489 IF (.NOT.resume) CALL states%write (times=times, env=envir) ! => outsv (module onf)
00490 DEALLOCATE (ydot)
00491 RETURN
00492 900 CONTINUE
00493 WRITE (stderr, "("&"plankton_init: ",A)") trim(iomsg)
00494 stop 1
00495 CONTAINS
00497 FUNCTION lfn (fn) RESULT (ix)
00498     IMPLICIT NONE
00499     CHARACTER (LEN=*) , INTENT(IN) :: fn
00500     INTEGER :: ix
00501     ix = scan(fn, '.', back=.true.) - 1 ! (position of last . in fn) - 1
00502     IF (ix.LE.0) ix = len_trim(fn)
00503 END FUNCTION lfn
00504 SUBROUTINE set_community (lun, lunpar)
00505     IMPLICIT NONE
00506     INTEGER, INTENT(IN) :: lun, lunpar
00507     CHARACTER (LEN=100), DIMENSION(100) :: dima="", dom(1)="", bact="", detr="", cmo="", cfo=""
00508     INTEGER :: mm, nn, n0
00509     LOGICAL :: coco_in_cf=.false.
00510     namelist /coco/ dima, dom, bact, detr, cmo, cfo
00511     rewind(unit=lun)
00512     READ (unit=lun, nml=coco, iomsg=iomsg, END=10, ERR=900)
00513     coco_in_cf = .true.
00514 10 CONTINUE
00515     IF (.NOT.coco_in_cf) THEN
00516         rewind(unit=lunpar)

```

```

00517     READ (unit=lunpar, nml=coco, iomsg=iomsg, err=900)
00518 END IF
00519 ! dima, detr, etc. are vectors of component names for each functional type
00520 envir%ndm = count(dima.NE.") ! dissolved matter types, e.g., NO3, PO4, O2, ...
00521 envir%ndom = count(dom.NE.") ! DOM types
00522 envir%ndet = count(detr.NE.") ! detritus types, e.g., small, large, ...
00523 envir%nbac = count(bact.NE.") ! bacteria types
00524 envir%nphy = count(cmo.NE.") ! phytoplankton types
00525 envir%nzoo = count(cfo.NE.") ! zooplankton types
00526 envir%nft = envir%ndom + envir%ndet + envir%nbac + envir%nphy + envir%nzoo
00527 ALLOCATE (envir%ipty(envir%nphy), envir%ipoc(envir%nft), fungrp(envir%nft), doma(envir%ndom), &
00528     detrit(envir%ndet), bacpla(envir%nbac), phypla(envir%nphy), zoopla(envir%nzoo))
00529 ! vector fungrp represents the community composition; it is a derived-type vector
00530 ! with just one component, var, which is a pointer to a specific functional group
00531 ! associate pointers fungrp(n)%var with individual functional groups
00532 DO nn = 1, envir%ndom ! DOM groups
00533     fungrp(nn)%var => doma(nn)
00534     fungrp(nn)%var$name = dom(nn)
00535 END DO
00536 n0 = envir%ndom
00537 DO nn = 1, envir%ndet ! detritus groups (small, large, ...)
00538     fungrp(n0+nn)%var => detrit(nn)
00539     fungrp(n0+nn)%var$name = detr(nn)
00540     IF (detrit(nn)%aggregates) envir%idagg = nn
00541 END DO
00542 IF (count(detrit%aggregates).GT.1) &
00543     stop 'There can be only one aggregates group.'
00544 IF ((envir%ndet.EQ.1).AND.(envir%idagg.EQ.0)) envir%idagg = 1
00545 n0 = n0 + envir%ndet
00546 DO nn=1, envir%nbac ! bacteria groups
00547     fungrp(n0+nn)%var => bacpla(nn)
00548     fungrp(n0+nn)%var$name = bact(nn)
00549 END DO
00550 n0 = n0 + envir%nbac
00551 DO nn=1, envir%nphy ! phytoplankton (cmo) groups
00552     envir%ipty(nn) = n0 + nn
00553     fungrp(n0+nn)%var => phypla(nn)
00554     fungrp(n0+nn)%var$name = cmo(nn)
00555 END DO
00556 n0 = n0 + envir%nphy
00557 DO nn=1, envir%nzoo ! zooplankton (cfo) groups
00558     fungrp(n0+nn)%var => zoopla(nn)
00559     fungrp(n0+nn)%var$name = cfo(nn)
00560 END DO
00561 n0 = n0 + envir%nzoo
00562 states%nsv_one_box = envir%ndm ! number of states in each box
00563 envir%nrs = 0 ! number of ratios across all functional groups
00564 envir%nagg = 0
00565 ! read parameters from namelists for all functional types (groups)
00566 DO nn = 1, envir%nft ! number of functional types
00567     fungrp(nn)%var%ifg = nn ! index of functional group
00568     rewind(unit=lunp)
00569     ! fungrp(nn)%var%read points to the read procedure pointer in each functional group
00570     ! for example, if fungrp(nn)%var is of TYPE(phycmo) (defined in module cmo), then
00571     ! fungrp(nn)%var%read => phypla%read => cmo_read, defined in module cmo
00572     CALL fungrp(nn)%var%read(env=envir, lun=lunp) ! read group parameters
00573     fungrp(nn)%var%ncon = SIZE(fungrp(nn)%var%nsv, fungrp(nn)%var%nnames(fungrp(nn)%var%nsv), &
00574         fungrp(nn)%var%units(fungrp(nn)%var%nsv), fungrp(nn)%var%icon(fungrp(nn)%var%ncon), &
00575         fungrp(nn)%var%ir(fungrp(nn)%var%nsv - 1))
00576     DO mm=1, fungrp(nn)%var%ncon ! set names, units, indices for constituents (C, N, ...) in group
00577         fungrp(nn)%var%icon(mm:mm) = pack([(k, k=1, envir%ncon)], &
00578             mask=fungrp(nn)%var%constituents(mm).EQ.constituents)
00579         fungrp(nn)%var%names(mm) &
00580             = trim(fungrp(nn)%var$name)//'_ '//trim(fungrp(nn)%var%constituents(mm))
00581         fungrp(nn)%var%units(mm) = trim(constituent_units(fungrp(nn)%var%icon(mm)))
00582     END DO
00583     fungrp(nn)%var%isv = states%nsv_one_box + [(mm, mm=1, fungrp(nn)%var%nsv)] ! state-variable
00584 indices
00585     envir%ipoc(nn) = fungrp(nn)%var%isv(1) ! index of biomass (C) of group
00586     envir%inq0 = envir%inq0 + fungrp(nn)%var%inq0 ! number of subsistence quotas across groups
00587     envir%nrs = envir%nrs + fungrp(nn)%var%nsv - 1 ! number of ratios (quotas & tracer-trait
products)
00588     states%nsv_one_box = states%nsv_one_box + fungrp(nn)%var%nsv ! number of state variables
00589     ! nagg is the number of types (groups) involved in aggregation
00590     IF (fungrp(nn)%var%sticky.GT.0d0) envir%nagg = envir%nagg + 1
00591 END DO
00592 ALLOCATE (states%var(states%nsv_one_box))
00593 states%var(1:envir%ndm)%name = dima(1:envir%ndm)
00594 DO nn = 1, envir%ndm ! envir%ndm is number of dissolved-matter states
00595     SELECT CASE (trim(dima(nn)))
00596     CASE ('CO2', 'DIC')
00597         envir%idic = nn
00598         states%var(nn)%units = 'mmolC m-3'
00599     CASE ('alkalinity')
00600         envir%ialk = nn
00601         states%var(nn)%units = 'mmol m-3'

```

```

00602     CASE ('O2')
00603         env%io2 = nn
00604         states%var(nn)%units = 'mmol m-3'
00605     CASE ('SiO3')
00606         env%isi = nn
00607         states%var(nn)%units = 'mmol m-3'
00608     CASE ('DIN', 'NO3')
00609         env%idin = nn
00610         states%var(nn)%units = 'mmolN m-3'
00611     CASE ('DIP', 'PO4')
00612         env%idip = nn
00613         states%var(nn)%units = 'mmolP m-3'
00614     CASE ('rDOC')
00615         env%irdoc = nn
00616         states%var(nn)%units = 'mmolC m-3'
00617     CASE ('rDON')
00618         env%irdon = nn
00619         states%var(nn)%units = 'mmolN m-3'
00620     CASE DEFAULT
00621         WRITE (stderr, '("Variable name ",A," is not implemented.")') trim(dima(nn))
00622         stop
00623     END SELECT
00624 END DO
00625 IF (env%idic.EQ.0) stop 'DIC missing in namelist coco' ! DIC and O2 are required
00626 IF (env%io2.EQ.0) stop 'O2 missing in namelist coco'
00627 IF ((env%ndom.GT.0).AND.(env%irdoc*env%irdon.EQ.0)) &
00628     stop 'DOM requires "rDOC" and "rDON" in dima'
00629 RETURN
00630 900 CONTINUE
00631 WRITE (stderr, '("set_community: ",A)') trim(iomsg)
00632 stop 1
00633 END SUBROUTINE set_community
00634 SUBROUTINE set_environment (lun)
00635     IMPLICIT NONE
00636     INTEGER, INTENT(IN) :: lun
00637     TYPE(quantity) :: PAR, wind, atmprs, CO2, temperature, salinity, ice,&
00638         potalk, totalalk,& !< potential, total surface alkalinity
00639         vdc, velocity, lacw, lachl, lapon, kag,&
00640         RQ,& !< respiratory quotient
00641         ptl
00642     CHARACTER(LEN=LEN(env%dicy)) :: dicy
00643     CHARACTER(LEN=NF90_MAX_NAME) :: daylength='Brock81'
00644     REAL(dp) :: daylen=0.5_dp, latdeg=0._dp, londeg=0._dp
00645     namelist /envi/ par, daylen, dicy, latdeg, londeg, wind, atmprs, co2, bottom, fsfalk,&
00646         temperature, salinity, ice, potalk, totalalk, vdc, velocity, lacw, lachl,&
00647         lapon, rq, nbfish, daylength, ptl, kag
00648     ! set default values and units for namelist envi
00649     ! allocation and assignments for env%daypar, env%wind, etc. are overridden in
00650     ! subroutine inpd (module onf) for those driven by surface datasets
00651     ALLOCATE (env%daypar, env%wind, env%pressure, env%xCO2, env%ice, env%lacw,&
00652         env%bbc(states%nsv_one_box), bottom(states%nsv_one_box), fsfalk(states%nsv_one_box),&
00653         position(states%nsv_one_box))
00654     par = quantity(0d0, 'W m-2') ! surface irradiance
00655     wind = quantity(0d0, 'm s-1') ! wind speed at 10 m
00656     ice = quantity(0d0, '1') ! ice cover fraction
00657     atmprs = quantity(1013.25d0, 'hPa') ! surface atmospheric pressure
00658     co2 = quantity(390d0, 'ppm') ! CO2 concentration in air
00659     temperature = quantity(10d0, '°C') ! surface air temperature
00660     vdc = quantity(0d0, 'm2 s-1') ! vertical eddy-diffusion coefficient
00661     velocity = quantity(0d0, 'm s-1') ! vertical velocity
00662     lacw = quantity(4d-2, 'm-1') ! light-attenuation coefficient of seawater
00663     salinity = quantity(33, 'psu')
00664     potalk = quantity(0d0, 'mmol m-3') ! potential alkalinity
00665     totalalk = quantity(0d0, 'mmol m-3') ! total alkalinity
00666     lachl = quantity(env%lachl, 'm2 gChl-1') ! light-attenuation coefficient of chl
00667     lapon = quantity(env%lapon, 'm2 molN-1') ! light-attenuation coefficient of PON
00668     rq = quantity(env%RQ, 'mol mol-1') ! respiratory quotient
00669     dicy = env%dicy ! diurnal light cycle
00670     ptl = quantity(times%ptl, 'degrees')
00671     bottom = bc(" , quantity(0._dp, ""))
00672     fsfalk = bc(" , quantity(0._dp, ""))
00673     position = (/ (n, n=1, states%nsv_one_box) /)
00674     env%bbc = 0d0 ! bottom-boundary conditions
00675     rewind(unit=lun)
00676     READ (unit=lun, nml=envi, iomsg=iomsg, err=900)
00677     env%dayPAR = convert(par, 'W m-2')
00678     env%wind = convert(wind, 'm s-1')
00679     env%ice = convert(ice, '1')
00680     env%pressure = convert(atmprs, 'Pa')
00681     IF (detrit(env%idagg)%kag.LT.0._dp) detrit(env%idagg)%kag = convert(kag, 's-1')
00682     dics%potalk = convert(potalk, 'mmol m-3')/35d0
00683     dics%totalalk = convert(totalalk, 'mmol m-3')
00684     env%xCO2 = convert(co2, "") ! convert ppmv to mole fraction
00685     env%RQ = convert(rq, 'mol mol-1')
00686     env%lacw = convert(lacw, 'm-1')
00687     env%lachl = convert(lachl, 'm2 mgChl-1')
00688     env%lapon = convert(lapon, 'm2 mmolN-1')

```

```

00689     envir%daylen => times%daylen
00690     envir%daylen = daylen
00691     envir%temperature = convert(temperature, 'C')
00692     envir%salinity = convert(salinity, 'psu')
00693     envir%vdc = convert(vdc, 'm2 s-1')
00694     envir%velocity = convert(velocity, 'm s-1')
00695     times%latdeg = latdeg      ! latitude in degrees N
00696     times%londeg = londeg      ! longitude in degrees E
00697     times%ptl = convert(ptl, 'rad')
00698     envir%dicy = dicy
00699     envir%daylenfun = daylength
00700     RETURN
00701 900 CONTINUE
00702     WRITE (stderr, '("set_environment: ",A)') trim(iomsg)
00703     stop 1
00704 END SUBROUTINE set_environment
00705 SUBROUTINE set_timing (lun)
00706     IMPLICIT NONE
00707     INTEGER, INTENT(IN) :: lun
00708     TYPE(quantity) :: dto,&      !< output time step
00709         start, end, sampling, timeout,&
00710         tin
00711     namelist /timing/ dto, start, end, sampling, timeout, tin
00712     dto = quantity(times%dto, 'h')      ! output time step
00713     start = quantity(times%start, 'd') ! start time relative to start of forcing file
00714     end = quantity(times%end, 'd')      ! end time relative to start of forcing file
00715     tin = quantity(times%tin, 'd')      ! time of initial conditions
00716     sampling = quantity(times%sample, 'h') ! sampling frequency
00717     timeout = quantity(real(times%timeout, dp), 'h') ! timeout (in case the integration hangs)
00718     rewind(unit=lun)
00719     READ (unit=lun, nml=timing, iomsg=iomsg, err=900)
00720     times%dto = convert(dto, 's')
00721     times%start = convert(start, 's')
00722     times%end = convert(end, 's')
00723     times%tin = convert(tin, 's')
00724     times%sample = convert(sampling, 's')
00725     times%timeout = int(convert(timeout, 's'))
00726     IF (times%londeg.GT.180) times%londeg = times%londeg - 360d0
00727     times%dtutc = nint(cfds*times%londeg)/360 ! difference between UTC and local time in s
00728     RETURN
00729 900 CONTINUE
00730     WRITE (stderr, '("set_timing: ",A)') trim(iomsg)
00731     stop 1
00732 END SUBROUTINE set_timing
00733 FUNCTION expand_path (file) RESULT (path)
00734     IMPLICIT NONE
00735     CHARACTER (LEN=NF90_MAX_NAME), INTENT(IN) :: file
00736     CHARACTER (LEN=NF90_MAX_NAME) :: path, ev, evn
00737     INTEGER :: status=0
00738     SELECT CASE (file(1:1))
00739     CASE ("~")
00740         evn = "HOME"
00741         CALL get_environment_variable (name=evn, VALUE=ev, status=status)
00742         path = trim(ev)//file(2:len_trim(file))
00743     CASE ("$$")
00744         pos = scan(file, "/")
00745         IF (file(2:2).EQ."{") THEN
00746             evn = file(3:pos-2)
00747         ELSE
00748             evn = file(2:pos-1)
00749         END IF
00750         CALL get_environment_variable (name=evn, VALUE=ev, status=status)
00751         path = trim(ev)//file(pos:len_trim(file))
00752     CASE DEFAULT
00753         path = file
00754     END SELECT
00755     IF (status.EQ.0) RETURN
00756     SELECT CASE (status)
00757     CASE (-1)
00758         WRITE (stderr, ' (A, " is too long.")') trim(evn)
00759         stop
00760     CASE (1)
00761         WRITE (stderr, ' (A, " is not set.")') trim(evn)
00762         stop
00763     CASE (2)
00764         stop 'Environment variables are not supported.'
00765     END SELECT
00766 END FUNCTION expand_path
00767 END SUBROUTINE plankton_init
00768
00770 SUBROUTINE lch_chl (ambient, boxes, stv)
00771     IMPLICIT NONE
00772     CLASS(local), INTENT(IN) :: ambient
00773     TYPE(layer), INTENT(INOUT) :: boxes(:)
00774     REAL(dp), INTENT(IN) :: stv(:, :)
00775     boxes%lch = (ambient%lacw + ambient%lachl*sum(stv(ambient%ichl,:), dim=1))*boxes%height
00776 END SUBROUTINE lch_chl

```

```

00778 SUBROUTINE lch_pon (ambient, boxes, stv)
00779   IMPLICIT NONE
00780   CLASS(local), INTENT(IN) :: ambient
00781   TYPE(layer), INTENT(INOUT) :: boxes(:)
00782   REAL(dp), INTENT(IN) :: stv(:, :)
00783   boxes%lch = (ambient%lacr + ambient%lapon*sum(stv(ambient%iphy,:), dim=1))*boxes%height
00784 END SUBROUTINE lch_pon
00785
00786 ! fraction of par arriving at box bottom
00787 SUBROUTINE fpar_vertical (env, boxes, nbox)
00788   CLASS(local), INTENT(INOUT) :: env
00789   TYPE(layer), INTENT(INOUT) :: boxes(:)
00790   INTEGER, INTENT(IN) :: nbox
00791   INTEGER :: nb
00792   DO nb=1,nbox
00793     env%fPAR(nb) = env%fPAR(nb-1)*exp(-boxes(nb)%lch)
00794     boxes(nb)%dPAR = env%dayPAR*(env%fPAR(nb-1) - env%fPAR(nb))/boxes(nb)%lch
00795   END DO
00796 END SUBROUTINE fpar_vertical
00797
00802 SUBROUTINE aggregate (env, grps, thisbox)
00803   IMPLICIT NONE
00804   CLASS(local), INTENT(INOUT) :: env
00805   TYPE(state), INTENT(IN) :: grps(:)
00806   TYPE(layer), INTENT(INOUT) :: thisbox
00807   INTEGER :: n
00808   thisbox%soma(0,detrit(env%idagg)%isv) = 0d0
00809   env%agg = env%sticky*matmul(env%OC(env%iagg)*env%sticky, env%kij)
00810   DO n=1,env%nagg
00811     thisbox%soma(0,grps(env%iagg(n))%var%isv) = -env%agg(n)*thisbox%stv(grps(env%iagg(n))%var%isv)
00812   END DO
00813   thisbox%soma(0,detrit(env%idagg)%isv) = thisbox%soma(0,detrit(env%idagg)%isv)&
00814     + matmul(env%quotas(detrit(env%idagg)%icon,env%iagg), env%OC(env%iagg)*env%agg)
00815   entry no_aggregate(env, grps, thisbox)
00816 END SUBROUTINE aggregate
00817
00818 SUBROUTINE alkalinity (env, thisbox)
00819   IMPLICIT NONE
00820   CLASS(local), INTENT(IN) :: env
00821   TYPE(layer), INTENT(INOUT) :: thisbox
00822   ! air-sea exchange is stored in fluxes(:,0), so does not affect alkalinity in soma
00823   thisbox%soma(:,env%ialk) = matmul(thisbox%soma(:,env%ifalk), env%fALK)
00824   entry no_alkalinity(env, thisbox)
00825 END SUBROUTINE alkalinity
00826
00833 SUBROUTINE boxbc_cd (env, boxes)
00834   IMPLICIT NONE
00835   CLASS(local), INTENT(IN) :: env
00836   TYPE(layer), INTENT(INOUT) :: boxes(:)
00837   WHERE (nint(sign(1._dp, env%dydz)).EQ.nint(sign(1._dp, env%vstv))) ! arithmetic means
00838     boxes(1)%bbc = (boxes(1)%stv - boxes(2)%stv)*boxes(1)%rdl + boxes(2)%stv
00839   ELSEWHERE
00840     ! geometric means:  $x^a y^{1-a} = \exp(a \cdot \log(x) + (1-a) \cdot \log(y))$ 
00841     boxes(1)%bbc = exp(boxes(1)%rdl*log(boxes(1)%stv) + (1._dp - boxes(1)%rdl)*log(boxes(2)%stv))
00842   END WHERE
00843   entry boxbc_dummy(env, boxes)
00844 END SUBROUTINE boxbc_cd
00845
00846 SUBROUTINE boxbc_upwind (env, boxes)
00847   IMPLICIT NONE
00848   CLASS(local), INTENT(IN) :: env
00849   TYPE(layer), INTENT(INOUT) :: boxes(:)
00850   WHERE (env%vstv.GT.0d0)
00851     boxes(1)%bbc = boxes(1)%stv
00852   ELSEWHERE
00853     boxes(1)%bbc = boxes(2)%stv
00854   END WHERE
00855 END SUBROUTINE boxbc_upwind
00856
00862 SUBROUTINE plankton_ode (neq, time, y, ydot)
00863   IMPLICIT NONE
00864   INTEGER, INTENT(IN) :: neq
00865   REAL(dp), INTENT(IN) :: time
00866   REAL(dp), INTENT(INOUT), TARGET :: y(*)
00867   REAL(dp), INTENT(OUT), TARGET :: ydot(*)
00868   REAL(dp) :: delvel
00869   INTEGER :: n, nb
00870   states%roc(1:states%nsv_one_box,1:phys%nbox) => ydot(1:states%nsv_all_boxes)
00871   IF (any(states%data.LT.0._dp)) THEN
00872     ydot(1:neq) = 0._dp
00873     RETURN
00874   END IF
00875   times%current = nint(time, kind=int64)
00876   times%hour = modulo(times%current + times%tbase, cfd)/3.6e3_dp
00877   times%toy = times%doy + times%hour/24._dp ! time of year in decimal days
00878   CALL phys%get (time) ! onf:readpd or onf:nopd -> get mixing, temperature, light, etc.
00879   CALL env%PARfun (times) ! PAR_const, PAR_brock81, PAR_MesoAqua, or PAR_LD
00880   CALL env%attenuation (boxes=box(1:phys%nbox), stv=states%data) ! lch_chl or lch_pon

```



```

00881     CALL envir%PARfrac (boxes=box(1:phys%nbox), nbox=phys%nbox)
00882     envir%PAR = envir%PAR*envir%fPAR      ! light intensity at box boundaries
00883     box(0)%flux(phys%isflx) = phys%surflux ! same as envir%fluxes(phys%isflx,0)
00884     IF (envir%xsflx) box(0)%flux(envir%ialk) = sum(box(0)%flux(envir%isfALK)*envir%fsfALK)
00885     box(phys%nbox+1)%stv(phys%ibot) = phys%botboc
00886     envir%all_ratios = states%data(envir%irt,:)/max(states%data(envir%irc,:), 1.e-30_dp) ! ratios
(including quotas)
00887     DO nb=1,phys%nbox
00888         ! depth-averages of instantaneous (iPAR) and average daytime (dPAR) irradiance in each box
00889         box(nb)%iPAR = (box(nb)%tPAR - box(nb)%bPAR)/box(nb)%lch
00890         CALL dics%alkalinity (box=box(nb)) ! set alkalinity
00891         CALL dics%co2 (box=box(nb), niter=dics%niter) ! calculate CO2, HCO3, CO3
00892         ! load ratios, quotas, POC (envir%OC) for current box (layer)
00893         envir%ratios = box(nb)%ratios
00894         envir%ratios(envir%iqr) = envir%ratios(envir%iqr) + envir%Q0
00895         ! subsistence N, P are removed to improve model stability (see Christian 2007)
00896         ! the CONSTRAINED option (see oppla.F90) does not prevent VODE_F90 from
00897         ! passing negative values in y
00898         envir%quotas = unpack(envir%ratios, mask=envir%qmsk, field=envir%quotas)
00899         envir%OC = states%data(envir%ipoc,nb) ! POC of all groups
00900         envir%dydz = (box(nb)%stv - box(nb+1)%stv)/box(nb)%dz ! vertical gradients
00901         ! fungrp(n)%var%flux points to the flux procedure pointer in each functional group
00902         ! for example, if fungrp(n)%var is of TYPE(phycmo) (defined in module cmo), then
00903         ! fungrp(n)%var%flux => phycmo%flux => cmo_flux, defined in module cmo
00904         DO n = 1,envir%nft
00905             CALL fungrp(n)%var%flux (grps=fungrp, env=envir, box=box(nb-1:nb+1), times=times)
00906         END DO
00907         CALL envir%aggregate (grps=fungrp, box=box(nb)) ! aggregate or no_aggregate
00908         ! physical transport: vertical advection and diffusion (mixing)
00909         delvel = box(nb-1)%velocity - box(nb)%velocity ! needed to correct for convergence
00910         envir%vvstv = box(nb)%velocity + envir%vvs*box(nb)%botperm ! advection + sinking + SVM + DVM +
...
00911     CALL envir%boxbc (boxes=box(nb:nb+1)) ! tracer concentrations at box boundaries
00912     ! flux: vertical flux out of (bottom of) box (advection + convergence + mixing)
00913     box(nb)%flux = box(nb)%bbc*envir%vvstv + delvel*states%data(:,nb) + envir%dydz*box(nb)%vdc
00914     CALL envir%alkalinity (thisbox=box(nb))
00915     END DO
00916     CALL dics%asf (env=envir, box=box(1)) ! air-sea fluxes of CO2 and O2
00917     states%roc = sum(envir%soma, dim=1) &
00918         + (envir%fluxes(:,0:phys%nbox-1) - envir%fluxes(:,1:phys%nbox))/envir%heights
00919     IF (states%write_soma) &
00920         ydot(states%i0flux:neq) = (/pack(envir%fluxes, mask=envir%fmsk), pack(envir%soma,
mask=envir%smask)/)
00921     END SUBROUTINE plankton_ode
00922
00923 END MODULE plankton

```

7.31 /Users/mpahlow/oppla/src/stdunt.f90 File Reference

Modules

- module `stdunt`
common parameters for oppla (dp, stdin, stdout, stderr)

Variables

- integer, parameter `stdunt::dp` =KIND(0D0)
- integer, parameter `stdunt::stderr` =0
- integer, parameter `stdunt::stdin` =5
- integer, parameter `stdunt::stdout` =6

7.32 stdunt.f90

[Go to the documentation of this file.](#)

```

00001
00002 MODULE stdunt
00003     IMPLICIT NONE
00004     INTEGER, PARAMETER :: dp=kind(0d0), stderr=0, stdin=5, stdout=6
00005 END MODULE stdunt

```


Bibliography

- Brock, T. D. (1981), Calculating solar radiation for ecological studies, *Ecol. Model.*, 14(1–2), 1–19. [13](#), [14](#), [15](#), [16](#), [19](#), [251](#)
- Dickson, A. G., and C. Goyet (1997), *DOE Handbook of Methods*, ORNL/CDIAC, vol. 74, Oak Ridge National Laboratory, Oak Ridge, Tennessee, version 2.1x. [52](#), [58](#)
- Follows, M. J., T. Ito, and S. Dutkiewicz (2006), On the solution of the carbonate chemistry system in ocean biogeochemistry models, *Ocean Model.*, 12, 290–301, DOI: [10.1016/j.ocemod.2005.05.004](#). [44](#), [53](#), [56](#), [273](#)
- Forsythe, W. C., E. J. Rykiel, R. S. Stahl, H.-i. Wu, and R. M. Schoolfield (1995), A model comparison for daylength as a function of latitude and day of year, *Ecol. Model.*, 80(1), 87–95, DOI: [10.1016/0304-3800\(94\)00034-f](#). [14](#), [15](#), [19](#)
- Garcia, H. E., and L. I. Gordon (1992), Oxygen solubility in seawater: Better fitting equations, *Limnol. Oceanogr.*, 37(6), 1307–1312, DOI: [10.4319/lo.1992.37.6.1307](#). [47](#), [153](#)
- Lewandowska, A., and U. Sommer (2010), Climate change and the spring bloom: a mesocosm study on the influence of light and temperature on phytoplankton and mesozooplankton, *Mar. Ecol. Prog. Ser.*, 405, 101–111, DOI: [10.3354/meps08520](#). [21](#)
- Liss, P. S., and L. Merlivat (1986), Air-sea gas exchange rates, in *The Role of Air-Sea Exchange in Geochemical Cycling*, NATO ASI Series C: Mathematical and Physical Sciences, vol. 185, edited by P. Buat-Ménard, pp. 113–129, Reidel, Dordrecht. [45](#), [57](#), [68](#), [154](#), [274](#)
- Millero, F. J. (1995), Thermodynamics of the carbon dioxide system in the oceans, *Geochim. Cosmochim. Acta*, 59, 661–677, DOI: [10.1016/0016-7037\(94\)00354-O](#). [44](#), [48](#), [59](#), [60](#), [154](#), [273](#)
- Millero, F. J., C.-T. Chen, A. Bradshaw, and K. Schleicher (1980), A new high pressure equation of state for seawater, *Deep-Sea Res.*, 27(3–4), 255–264, DOI: [10.1016/0198-0149\(80\)90016-3](#). [44](#), [52](#), [273](#)
- Munhoven, G. (2013), Mathematics of the total alkalinity-pH equation — pathway to robust and universal solution algorithms: the SolveSAPHE package v1.0.1, *Geosci. Model Dev.*, 6, 1367–1388, DOI: [10.5194/gmd-6-1367-2013](#). [44](#), [45](#), [54](#), [56](#), [273](#)
- Pahlow, M., and A. E. F. Prowe (2010), Model of optimal current feeding in zooplankton, *Mar. Ecol. Prog. Ser.*, 403, 129–144, DOI: [10.3354/meps08466](#). [24](#)
- Pahlow, M., H. Dietze, and A. Oschlies (2013), Optimality-based model of phytoplankton growth and diazotrophy, *Mar. Ecol. Prog. Ser.*, 489, 1–16, DOI: [10.3354/meps10449](#). [34](#)
- Wanninkhof, R. (1992), Relationship between wind speed and gas exchange over the ocean, *J. Geophys. Res.*, 97(C5), 7373–7382, DOI: [10.1029/92JC00188](#). [45](#), [57](#), [69](#), [154](#), [274](#)
- Wanninkhof, R. (2014), Relationship between wind speed and gas exchange over the ocean revisited, *Limnol. Oceanogr. Methods*, 12, 351–362, DOI: [10.4319/lom.2014.12.351](#). [45](#), [57](#), [68](#), [154](#), [274](#)
- Wanninkhof, R., and W. R. McGillis (1999), A cubic relationship between air-sea CO₂ exchange and wind speed, *Geophys. Res. Lett.*, 26(13), 1889–1892. [45](#), [57](#), [70](#), [154](#), [274](#)

Index

/Users/mpahlow/oppla/src/bac.f90, [247](#), [248](#)
/Users/mpahlow/oppla/src/brock81.f90, [251](#), [252](#)
/Users/mpahlow/oppla/src/cfo.f90, [254](#), [255](#)
/Users/mpahlow/oppla/src/clops.f90, [263](#)
/Users/mpahlow/oppla/src/cmo.f90, [264](#), [265](#)
/Users/mpahlow/oppla/src/det.f90, [270](#)
/Users/mpahlow/oppla/src/dic.f90, [273](#), [277](#)
/Users/mpahlow/oppla/src/dvode.f90, [284](#), [285](#)
/Users/mpahlow/oppla/src/et.f90, [286](#), [292](#)
/Users/mpahlow/oppla/src/fudu.F90, [296](#), [297](#)
/Users/mpahlow/oppla/src/julian.f90, [299](#), [302](#)
/Users/mpahlow/oppla/src/lambert.f90, [306](#), [307](#)
/Users/mpahlow/oppla/src/onf.f90, [312](#), [314](#)
/Users/mpahlow/oppla/src/oppla.F90, [324](#), [326](#)
/Users/mpahlow/oppla/src/plankton.f90, [328](#), [332](#)
/Users/mpahlow/oppla/src/stdunt.f90, [342](#)

a
 cmo::phycmo, [197](#)

a0
 cmo::phycmo, [197](#)

ac
 dic::dicsys, [155](#)

acd
 dic::dicsys, [155](#)

adim
 bac::bacteria, [136](#)

adop
 bac::bacteria, [136](#)

advect
 onf::phydat, [209](#)

af
 cfo::zoocfo, [230](#)

aggregate
 plankton, [121](#)

aggregates
 det::detritus, [148](#)

aggregation
 et::aggregation, [131](#)

alkalinity
 dic::dicsys, [155](#)
 plankton, [122](#)

all_states
 onf::statevars, [215](#)

alpha
 cmo::phycmo, [197](#)

alphanat
 cmo::phycmo, [197](#)

ascii
 fudu, [85](#)

asf
 dic::dicsys, [153](#)

asf_o2
 dic.f90, [274](#)

asfco2
 dic, [46](#)
 dic::dicsys, [155](#)

asfo2
 dic::dicsys, [155](#)

at
 cfo::zoocfo, [230](#)
 dic::dicsys, [155](#)

atol
 dvode::ode, [188](#)

attenuate
 et::attenuate, [132](#)

attributes
 onf::nf90group, [180](#)

bac, [9](#)
 bac_flux, [10](#)
 bac_grow, [10](#)
 bac_read, [10](#)
 bac_set, [11](#)
 bac_uptake, [11](#)
 bac_uptake_ft, [11](#)
 dom_flux, [12](#)
 dom_read, [12](#)
 dom_set, [12](#)
 bac::bacteria, [132](#)
 adim, [136](#)
 adop, [136](#)
 doc, [136](#)
 docnp, [136](#)
 dom, [136](#)
 don, [136](#)
 dop, [136](#)
 flux, [135](#)
 ftem, [137](#)
 ggem, [137](#)
 grow, [137](#)
 idocnp, [137](#)
 idom, [137](#)
 lambda, [137](#)
 n2p, [137](#)
 qdon, [138](#)
 qdop, [138](#)
 qn_param, [138](#)
 qp_param, [138](#)
 rc, [138](#)

- read, 135
- set, 135
- uptake, 138
- vdin, 138
- vdip, 139
- vdoc, 139
- vdom, 139
- vdon, 139
- vdop, 139
- vm, 139
- vmax, 139
- zeta, 139
- bac::dom, 163
 - flux, 165
 - ldlc, 166
 - ldln, 166
 - read, 165
 - relac, 166
 - relan, 166
 - set, 165
 - trlc, 166
 - trln, 166
- bac_flux
 - bac, 10
- bac_grow
 - bac, 10
- bac_read
 - bac, 10
- bac_set
 - bac, 11
- bac_uptake
 - bac, 11
- bac_uptake_ft
 - bac, 11
- bacpla
 - plankton, 128
- beta
 - cfo::zoocfo, 230
- bfn
 - onf, 117
- botbc
 - onf::phydat, 209
- bottom
 - onf::phydat, 209
- bounds
 - onf::dimension, 162
 - onf::nf90group, 180
- box
 - plankton, 128
- boxalk
 - et::boxalk, 140
- boxbc
 - et::boxbc, 141
- boxbc_cd
 - plankton, 122
- boxbc_upwind
 - plankton, 123
- brock81, 13
 - brock81_init, 14
 - cfhds, 22
 - csrad, 14
 - d15, 22
 - daylength, 15
 - declination_b81, 16
 - declination_f95, 16
 - fd, 22
 - par_brock81, 17
 - par_ld, 18
 - par_meso aqua, 18
 - rd rad, 18
 - sdl, 22
 - solar, 22
 - sunday_brock81, 19
 - sunday_const, 20
 - sunday_ld, 20
 - sunday_meso aqua, 21
 - tilt, 22
 - year days, 23
- brock81_init
 - brock81, 14
- bt
 - dic::dicsys, 155
- c_strftime
 - julian::c_strftime, 141
- c_strptime
 - julian::c_strptime, 142
- ca
 - cfo::zoocfo, 230
 - dic::dicsys, 155
- calc
 - cmo::phycmo, 197
- catch_signal_2
 - oppla.F90, 325
- cds
 - onf::phydat, 209
- cf
 - cfo::zoocfo, 230
- cf0
 - cfo::zoocfo, 230
- cfds
 - fudu, 85
- cfhds
 - brock81, 22
- cfhs
 - fudu, 85
- cfo, 23
 - cfo_dvm, 24
 - cfo_flux, 25
 - cfo_read, 25
 - cfo_set, 26
 - cfo_svm_dyn, 26
 - cfo_svm_static, 27
 - egest_dm, 28
 - egest_pm, 28
 - excrete_dim, 29
 - excrete_dom, 29

- foract, [30](#)
- forage_cfo, [30](#)
- forage_switch, [31](#)
- cfo::pcc, [24](#), [255](#)
- cfo::zoocfo, [223](#)
 - af, [230](#)
 - at, [230](#)
 - beta, [230](#)
 - ca, [230](#)
 - cf, [230](#)
 - cf0, [230](#)
 - cmax, [231](#)
 - cxda, [231](#)
 - cxdd, [231](#)
 - d0dvm, [231](#)
 - d0svm, [231](#)
 - depdvm, [231](#)
 - depsvm, [232](#)
 - dodvm, [232](#)
 - dodvm0, [232](#)
 - dom, [232](#)
 - dosvm, [232](#)
 - doya, [232](#)
 - doyd, [233](#)
 - dtldvm, [233](#)
 - dtsvma, [233](#)
 - dtsvmd, [233](#)
 - dvmig, [233](#)
 - dynsvm, [233](#)
 - e, [234](#)
 - egest, [234](#)
 - egestion, [234](#)
 - emax, [234](#)
 - eq, [234](#)
 - excrete, [234](#)
 - fdchl, [235](#)
 - fde, [235](#)
 - fdpic, [235](#)
 - fhphi, [235](#)
 - fhrm, [235](#)
 - flux, [229](#)
 - forage, [235](#)
 - fpx, [236](#)
 - fsvm, [236](#)
 - g, [236](#)
 - gmax, [236](#)
 - i, [236](#)
 - icdet, [236](#)
 - id0svm, [237](#)
 - id0svm1, [237](#)
 - id1dvm, [237](#)
 - id1svm, [237](#)
 - id1svm1, [237](#)
 - ida, [237](#)
 - idd, [238](#)
 - idet, [238](#)
 - idig, [238](#)
 - idim, [238](#)
 - idom, [238](#)
 - imaf, [238](#)
 - imax, [238](#)
 - imot, [239](#)
 - inot, [239](#)
 - ip, [239](#)
 - irda, [239](#)
 - irdd, [239](#)
 - isda, [239](#)
 - isdd, [239](#)
 - mort, [240](#)
 - ngr, [240](#)
 - pcc, [240](#)
 - phi, [240](#)
 - phi0, [240](#)
 - phim, [240](#)
 - phin, [241](#)
 - pp, [241](#)
 - ppm, [241](#)
 - ppn, [241](#)
 - pq, [241](#)
 - pth, [241](#)
 - qn_param, [242](#)
 - qp_param, [242](#)
 - quotas, [242](#)
 - r, [242](#)
 - read, [229](#)
 - rff, [242](#)
 - rffm, [242](#)
 - rffn, [243](#)
 - rm, [243](#)
 - rm0, [243](#)
 - rq, [243](#)
 - set, [229](#)
 - svmig, [243](#)
 - switch, [243](#)
 - thcsvm, [244](#)
 - toy, [244](#)
 - v0svm, [244](#)
 - vdvm, [244](#)
 - vsvm, [244](#)
 - vsvm0, [244](#)
 - vsvm1, [245](#)
 - xq, [245](#)
- cfo_dvm
 - cfo, [24](#)
- cfo_flux
 - cfo, [25](#)
- cfo_read
 - cfo, [25](#)
- cfo_set
 - cfo, [26](#)
- cfo_svm_dyn
 - cfo, [26](#)
- cfo_svm_static
 - cfo, [27](#)
- chl
 - cmo::phycmo, [197](#)

- chl0
 - cmo::phycmo, 197
- chl_dyn
 - cmo, 34
- chldyn
 - cmo::phycmo, 198
- clops, 32
 - getopt, 32
- clower
 - dvode::ode, 188
- clsbot
 - onf::phydat, 209
- cmax
 - cfo::zoocfo, 231
- cmo, 33
 - chl_dyn, 34
 - cmo_flux, 34
 - cmo_pic, 34
 - cmo_read, 35
 - cmo_set, 36
 - facn2f, 36
 - ftnf_eppley, 37
 - ftnf_houlton, 37
 - ftnf_oppla, 38
 - grow, 38
 - tchdyn, 39
 - tchia, 40
 - uptake, 40
- cmo::phycmo, 191
 - a, 197
 - a0, 197
 - alpha, 197
 - alphan, 197
 - calc, 197
 - chl, 197
 - chl0, 197
 - chldyn, 198
 - daylen, 198
 - dlfa, 198
 - dom, 198
 - dynamicchl, 198
 - f0n, 198
 - fc, 199
 - fcddq, 199
 - ff, 199
 - flux, 196
 - fn, 199
 - fpic, 199
 - ftalpha, 199
 - ftemp, 200
 - ftnf, 200
 - ftnc, 200
 - fv, 200
 - growth, 200
 - ic, 200
 - ichl, 201
 - idoc, 201
 - ipic, 201
 - mu0, 201
 - n2f, 201
 - ngr, 201
 - nuts, 201
 - pachl, 202
 - par, 202
 - pic, 202
 - q0n, 202
 - q0p, 202
 - qpac, 202
 - qsn, 203
 - rc, 203
 - rct, 203
 - rctht, 203
 - rdl, 203
 - read, 196
 - set, 196
 - si, 203
 - svph, 204
 - tch, 204
 - theta, 204
 - v0, 204
 - vc, 204
 - vdic, 204
 - vdoc, 205
 - vn, 205
 - vp, 205
 - vph0, 205
 - vpic, 205
 - vsink, 205
 - zc, 206
 - zn, 206
 - znf, 206
 - znu, 206
- cmo_flux
 - cmo, 34
- cmo_pic
 - cmo, 34
- cmo_read
 - cmo, 35
- cmo_set
 - cmo, 36
- co2
 - dic, 47
 - dic::dicsys, 153
- constituent_units
 - et, 80
- constituents
 - et, 80
 - et::fungroup, 171
- constr
 - dvode::ode, 188
- constrained
 - dvode::ode, 188
- convert
 - fudu, 83
- count
 - onf::statevars, 215

csrad
 brock81, 14
ct
 dic::dicsys, 156
cupper
 dvode::ode, 188
cv_convert_double
 fudu::cv_convert, 143
cv_convert_float
 fudu::cv_convert, 143
cv_free
 fudu::cv_free, 143
cxda
 cfo::zoocfo, 231
cxdd
 cfo::zoocfo, 231

d0dvm
 cfo::zoocfo, 231
d0svm
 cfo::zoocfo, 231
d15
 brock81, 22
data
 onf::phydat, 210
 onf::statevars, 215
daylen
 cmo::phycmo, 198
daylength
 brock81, 15
decay
 det::detritus, 148
decl
 et::decl, 144
declination_b81
 brock81, 16
declination_f95
 brock81, 16
deltaalk
 dic.f90, 275
depdvm
 cfo::zoocfo, 231
depsvm
 cfo::zoocfo, 232
depth
 onf::nf90group, 180
depth_flux
 onf::nf90group, 181
det, 41
 det_flux, 42
 det_fluxes, 42
 det_loss_pic, 42
 det_read, 42
 det_set, 43
det::detritus, 144
 aggregates, 148
 decay, 148
 detchl, 148
 detpic, 148
 flux, 147
 fluxes, 148
 formpic, 149
 iloss, 149
 irec, 149
 isvl, 149
 kag, 149
 loss, 149
 losses, 150
 nrec, 150
 omax, 150
 read, 147
 remi, 150
 remic, 150
 remichl, 150
 remin, 150
 remip, 151
 remipic, 151
 set, 148
 sink, 151
det_flux
 det, 42
det_fluxes
 det, 42
det_loss_pic
 det, 42
det_read
 det, 42
det_set
 det, 43
detchl
 det::detritus, 148
detpic
 det::detritus, 148
detrit
 plankton, 128
dfan
 onf, 117
dic, 43
 asfco2, 46
 co2, 47
 dic_potalk, 49
 dicalp, 50
 eos80_rho, 52
 faco2, 52
 hini_f06, 53
 hini_m13, 54
 hsolve_f06, 55
 hsolve_m13, 56
 indic, 57
 k0co2, 58
 lgkspa, 59
 lgkspc, 59
 lnk1_sws, 60
 lnk1_total, 61
 lnk1p_sws, 61
 lnk2_sws, 62
 lnk2_total, 62

- lnk2p_sws, 63
- lnk3p_sws, 63
- lnkb_total, 64
- lnkf_free, 65
- lnkf_total, 65
- lnknh4_sws, 66
- lnks_free, 66
- lnkw_sws, 67
- pivel_lm86, 68
- pivel_w14, 68
- pivel_w92, 69
- pivel_w99, 70
- rgas, 70
- t0ck, 71
- dic.f90
 - asf_o2, 274
 - deltaalk, 275
 - hset_m13, 275
- dic::dicsys, 151
 - ac, 155
 - acd, 155
 - alkalinity, 155
 - asf, 153
 - asfco2, 155
 - asfo2, 155
 - at, 155
 - bt, 155
 - ca, 155
 - co2, 153
 - ct, 156
 - faco2, 156
 - ft, 156
 - gmk, 156
 - hsolve, 156
 - init, 156
 - k0co2, 156
 - k1, 156
 - k12, 157
 - k12p, 157
 - k1p, 157
 - k2, 157
 - k2p, 157
 - k3p, 157
 - kb, 157
 - kf, 157
 - knh4, 158
 - ksi, 158
 - kso4, 158
 - kspa, 158
 - kspc, 158
 - kw, 158
 - niter, 158
 - ph0, 158
 - pivel, 159
 - potalk, 159
 - pt, 159
 - r_tot_free, 159
 - r_tot_sws, 159
 - read, 154
 - rrnp, 159
 - rrnsi, 160
 - schnc02, 160
 - schno2, 160
 - sit, 160
 - so4, 160
 - tk, 160
 - totalk, 160
 - tvco2, 161
 - tvo2, 161
- dic_potalk
 - dic, 49
- dicalp
 - dic, 50
- dics
 - plankton, 128
- dlfa
 - cmo::phycmo, 198
- doc
 - bac::bacteria, 136
- docnp
 - bac::bacteria, 136
- dodvm
 - cfo::zoocfo, 232
- dodvm0
 - cfo::zoocfo, 232
- dom
 - bac::bacteria, 136
 - cfo::zoocfo, 232
 - cmo::phycmo, 198
- dom_flux
 - bac, 12
- dom_read
 - bac, 12
- dom_set
 - bac, 12
- doma
 - plankton, 128
- don
 - bac::bacteria, 136
- dop
 - bac::bacteria, 136
- dosvm
 - cfo::zoocfo, 232
- doya
 - cfo::zoocfo, 232
- doyd
 - cfo::zoocfo, 233
- dp
 - julian, 104
 - lambert, 108
 - stdunt, 130
- dtdvm
 - cfo::zoocfo, 233
- dtsvma
 - cfo::zoocfo, 233
- dtsvmd

- cfo::zoocfo, 233
- dvmig
 - cfo::zoocfo, 233
- dvode, 71
 - read_ode, 71
 - set_atol, 72
- dvode::ode, 186
 - atol, 188
 - clower, 188
 - constr, 188
 - constrained, 188
 - cupper, 188
 - h0, 188
 - hmax, 188
 - hmin, 189
 - itask, 189
 - jsv, 189
 - maxord, 189
 - meth, 189
 - mf, 189
 - miter, 189
 - moss, 189
 - mxatol, 190
 - mxhnil, 190
 - mxrtol, 190
 - mxstep, 190
 - names, 190
 - nconstr, 190
 - neq, 190
 - quiet, 191
 - read, 187
 - rtol, 191
 - set_atol, 187
 - sparse_jacobian, 191
- dynamicchl
 - cmo::phycmo, 198
- dynsvm
 - cfo::zoocfo, 233
- e
 - cfo::zoocfo, 234
- egest
 - cfo::zoocfo, 234
- egest_dm
 - cfo, 28
- egest_pm
 - cfo, 28
- egestion
 - cfo::zoocfo, 234
- emax
 - cfo::zoocfo, 234
- envir
 - plankton, 128
- eos80_rho
 - dic, 52
- eq
 - cfo::zoocfo, 234
- et, 72
 - constituent_units, 80
- constituents, 80
 - ft_eppley, 78
 - ft_houlton, 78
 - ft_me, 78
 - ft_oppla, 79
 - ft_select, 79
 - pi, 80
 - rad, 81
- et::aggregation, 131
 - aggregation, 131
- et::attenuate, 131
 - attenuate, 132
- et::boxalk, 140
 - boxalk, 140
- et::boxbc, 140
 - boxbc, 141
- et::decl, 144
 - decl, 144
- et::fluxes, 167
 - fluxes, 167
- et::fpar, 167
 - fpar, 168
- et::fungroup, 168
 - constituents, 171
 - flux, 170
 - ft, 171
 - ft_select, 170
 - icon, 171
 - ifg, 171
 - iq0, 171
 - ir, 172
 - isv, 172
 - motile, 172
 - name, 172
 - names, 172
 - ncon, 172
 - nq0, 173
 - nsv, 173
 - oc, 173
 - q0, 173
 - q10, 173
 - qn, 173
 - qp, 174
 - ratios, 174
 - read, 170
 - set, 171
 - stick, 174
 - sticky, 174
 - topt, 174
 - tref, 174
 - tspr, 174
 - units, 175
 - vv, 175
- et::init, 175
 - init, 175
- et::layer, 73, 288
- et::light, 177
 - light, 177

et::local, 74, 289
 et::preset, 211
 preset, 212
 et::state, 76, 291
 et::sunday, 218
 sunday, 218
 et::timing, 77, 291
 excrete
 cfo::zoocfo, 234
 excrete_dim
 cfo, 29
 excrete_dom
 cfo, 29
 exof
 onf, 110
 expand_path
 plankton.f90, 330
 exudu
 fudu, 83

 f0n
 cmo::phycmo, 198
 facn2f
 cmo, 36
 faco2
 dic, 52
 dic::dicsys, 156
 fc
 cmo::phycmo, 199
 fcdtdq
 cmo::phycmo, 199
 fd
 brock81, 22
 fdchl
 cfo::zoocfo, 235
 fde
 cfo::zoocfo, 235
 fdpic
 cfo::zoocfo, 235
 ff
 cmo::phycmo, 199
 fhphi
 cfo::zoocfo, 235
 fhrm
 cfo::zoocfo, 235
 filename
 onf::nf90group, 181
 flux
 bac::bacteria, 135
 bac::dom, 165
 cfo::zoocfo, 229
 cmo::phycmo, 196
 det::detritus, 147
 et::fungroup, 170
 onf::statevars, 215
 flux_depth
 onf::statevars, 216
 fluxes
 det::detritus, 148
 et::fluxes, 167
 fn
 cmo::phycmo, 199
 foract
 cfo, 30
 forage
 cfo::zoocfo, 235
 forage_cfo
 cfo, 30
 forage_switch
 cfo, 31
 formpic
 det::detritus, 149
 fpar
 et::fpar, 168
 onf::phydat, 210
 fpar_vertical
 plankton, 123
 fpic
 cmo::phycmo, 199
 fpx
 cfo::zoocfo, 236
 fsm
 cfo::zoocfo, 236
 ft
 dic::dicsys, 156
 et::fungroup, 171
 ft_eppley
 et, 78
 ft_houlton
 et, 78
 ft_me
 et, 78
 ft_oppla
 et, 79
 ft_select
 et, 79
 et::fungroup, 170
 ftalpha
 cmo::phycmo, 199
 ftem
 bac::bacteria, 137
 ftemp
 cmo::phycmo, 200
 ftnf
 cmo::phycmo, 200
 ftnf_eppley
 cmo, 37
 ftnf_houlton
 cmo, 37
 ftnf_oppla
 cmo, 38
 ftrc
 cmo::phycmo, 200
 fudu, 81
 ascii, 85
 cfds, 85
 cfhs, 85

- convert, [83](#)
- exudu, [83](#)
- inudu, [83](#)
- latin1, [86](#)
- slot, [84](#)
- udurr, [85](#)
- ute_bad_arg, [86](#)
- ute_cant_format, [86](#)
- ute_exists, [86](#)
- ute_meaningless, [86](#)
- ute_no_second, [86](#)
- ute_no_unit, [86](#)
- ute_not_same_system, [86](#)
- ute_open_arg, [87](#)
- ute_open_default, [87](#)
- ute_open_env, [87](#)
- ute_os, [87](#)
- ute_parse, [87](#)
- ute_success, [87](#)
- ute_syntax, [87](#)
- ute_unknown, [87](#)
- ute_visit_error, [88](#)
- utencoding, [88](#)
- utf8, [88](#)
- utsystem, [88](#)
- fudu::cv_convert, [142](#)
 - cv_convert_double, [143](#)
 - cv_convert_float, [143](#)
- fudu::cv_free, [143](#)
 - cv_free, [143](#)
- fudu::quantity, [82](#), [297](#)
- fudu::ut_decode_time, [219](#)
 - ut_decode_time, [219](#)
- fudu::ut_encode_time, [219](#)
 - ut_encode_time, [220](#)
- fudu::ut_free, [220](#)
 - ut_free, [220](#)
- fudu::ut_free_system, [220](#)
 - ut_free_system, [221](#)
- fudu::ut_get_converter, [221](#)
 - ut_get_converter, [221](#)
- fudu::ut_get_status, [221](#)
 - ut_get_status, [222](#)
- fudu::ut_parse, [222](#)
 - ut_parse, [222](#)
- fudu::ut_read_xml, [223](#)
 - ut_read_xml, [223](#)
- fungrp
 - plankton, [129](#)
- fv
 - cmo::phycmo, [200](#)
- g
 - cfo::zoocfo, [236](#)
- get
 - onf::phydat, [210](#)
- getopt
 - clops, [32](#)
- ggem
 - bac::bacteria, [137](#)
- gmax
 - cfo::zoocfo, [236](#)
- gmk
 - dic::dicsys, [156](#)
- gregorian_day
 - julian, [104](#)
- gregorian_dutc
 - julian, [104](#)
- gregorian_month
 - julian, [104](#)
- gregorian_year
 - julian, [104](#)
- grow
 - bac::bacteria, [137](#)
 - cmo, [38](#)
- growth
 - cmo::phycmo, [200](#)
- h0
 - dvode::ode, [188](#)
- hini_f06
 - dic, [53](#)
- hini_m13
 - dic, [54](#)
- hmax
 - dvode::ode, [188](#)
- hmin
 - dvode::ode, [189](#)
- hset_m13
 - dic.f90, [275](#)
- hsolve
 - dic::dicsys, [156](#)
- hsolve_f06
 - dic, [55](#)
- hsolve_m13
 - dic, [56](#)
- i
 - cfo::zoocfo, [236](#)
- i0flux
 - onf::statevars, [216](#)
- i0sms
 - onf::statevars, [216](#)
- ibot
 - onf::phydat, [210](#)
- ic
 - cmo::phycmo, [200](#)
- icdet
 - cfo::zoocfo, [236](#)
- ichl
 - cmo::phycmo, [201](#)
- ico2ds
 - onf, [117](#)
- icon
 - et::fungroup, [171](#)
- id
 - onf::dimension, [162](#)
 - onf::nf90group, [181](#)

id0svm
 cfo::zoocfo, 237
 id0svm1
 cfo::zoocfo, 237
 id1dvm
 cfo::zoocfo, 237
 id1svm
 cfo::zoocfo, 237
 id1svm1
 cfo::zoocfo, 237
 ida
 cfo::zoocfo, 237
 idd
 cfo::zoocfo, 238
 idet
 cfo::zoocfo, 238
 idig
 cfo::zoocfo, 238
 idim
 cfo::zoocfo, 238
 idoc
 cmo::phycmo, 201
 idocnp
 bac::bacteria, 137
 idom
 bac::bacteria, 137
 cfo::zoocfo, 238
 ifg
 et::fungroup, 171
 ilat
 onf::nf90group, 181
 ilon
 onf::nf90group, 181
 iloss
 det::detritus, 149
 imaf
 cfo::zoocfo, 238
 imax
 cfo::zoocfo, 238
 imot
 cfo::zoocfo, 239
 indic
 dic, 57
 infan
 onf, 117
 info
 onf::nf90group, 180
 init
 dic::dicsys, 156
 et::init, 175
 onf::phydat, 209
 innf
 onf, 110
 inot
 cfo::zoocfo, 239
 inpd
 onf, 111
 inudu
 fudu, 83
 invar
 onf, 111
 ip
 cfo::zoocfo, 239
 ipic
 cmo::phycmo, 201
 iprsds
 onf, 117
 iq0
 et::fungroup, 171
 ir
 et::fungroup, 172
 irda
 cfo::zoocfo, 239
 irdd
 cfo::zoocfo, 239
 irec
 det::detritus, 149
 isda
 cfo::zoocfo, 239
 isdd
 cfo::zoocfo, 239
 isflx
 onf::phydat, 210
 isv
 et::fungroup, 172
 isvl
 det::detritus, 149
 it
 onf::statevars, 216
 it0
 onf::nf90group, 181
 itask
 dvote::ode, 189
 jd_of_j2000_noon
 julian, 105
 jdn_of_j2000
 julian, 105
 jsv
 dvote::ode, 189
 jul_dhmsofsec
 julian, 91
 jul_dsofsec
 julian, 92
 jul_dutcofynd
 julian, 92
 jul_et_type
 julian, 105
 jul_fixym
 julian, 93
 jul_formatdate
 julian, 93
 jul_gregymdofjd
 julian, 94
 jul_initleaps
 julian, 94
 jul_isleapday

- julian, [95](#)
- jul_jdofgregymd
 - julian, [95](#)
- jul_jdofjulymd
 - julian, [96](#)
- jul_julymdofjd
 - julian, [96](#)
- jul_leapsecs
 - julian, [97](#)
- jul_leapsecsym
 - julian, [97](#)
- jul_parsedt
 - julian, [98](#)
- jul_tai_type
 - julian, [105](#)
- jul_taiofdutc
 - julian, [99](#)
- jul_taiofet
 - julian, [100](#)
- jul_taiofjd
 - julian, [101](#)
- jul_utc_type
 - julian, [105](#)
- jul_ydofdutc
 - julian, [101](#)
- jul_ymdofdutc
 - julian, [102](#)
- julian, [88](#)
 - dp, [104](#)
 - gregorian_day, [104](#)
 - gregorian_dutc, [104](#)
 - gregorian_month, [104](#)
 - gregorian_year, [104](#)
 - jd_of_j2000_noon, [105](#)
 - jdn_of_j2000, [105](#)
 - jul_dhmsfsec, [91](#)
 - jul_dsofsec, [92](#)
 - jul_dutcofynd, [92](#)
 - jul_et_type, [105](#)
 - jul_fixym, [93](#)
 - jul_formatdate, [93](#)
 - jul_gregymdofjd, [94](#)
 - jul_initleaps, [94](#)
 - jul_isleapday, [95](#)
 - jul_jdofgregymd, [95](#)
 - jul_jdofjulymd, [96](#)
 - jul_julymdofjd, [96](#)
 - jul_leapsecs, [97](#)
 - jul_leapsecsym, [97](#)
 - jul_parsedt, [98](#)
 - jul_tai_type, [105](#)
 - jul_taiofdutc, [99](#)
 - jul_taiofet, [100](#)
 - jul_taiofjd, [101](#)
 - jul_utc_type, [105](#)
 - jul_ydofdutc, [101](#)
 - jul_ymdofdutc, [102](#)
 - leap_defaults, [105](#)
 - leap_index, [103](#)
 - leap_table, [106](#)
 - leaps, [106](#)
 - mjd_of_j2000_noon, [106](#)
- julian::c_strftime, [141](#)
 - c_strftime, [141](#)
- julian::c_strptime, [142](#)
 - c_strptime, [142](#)
- julian::leap, [90](#), [301](#)
- julian::tm_struct, [90](#), [301](#)
- julian::yms, [91](#), [301](#)
- k0co2
 - dic, [58](#)
 - dic::dicsys, [156](#)
- k1
 - dic::dicsys, [156](#)
- k12
 - dic::dicsys, [157](#)
- k12p
 - dic::dicsys, [157](#)
- k1p
 - dic::dicsys, [157](#)
- k2
 - dic::dicsys, [157](#)
- k2p
 - dic::dicsys, [157](#)
- k3p
 - dic::dicsys, [157](#)
- kag
 - det::detritus, [149](#)
- kb
 - dic::dicsys, [157](#)
- kf
 - dic::dicsys, [157](#)
- knh4
 - dic::dicsys, [158](#)
- ksi
 - dic::dicsys, [158](#)
- kso4
 - dic::dicsys, [158](#)
- kspa
 - dic::dicsys, [158](#)
- kspc
 - dic::dicsys, [158](#)
- kw
 - dic::dicsys, [158](#)
- lambda
 - bac::bacteria, [137](#)
- lambert, [106](#)
 - dp, [108](#)
 - lwm1, [107](#)
 - wapd, [107](#)
 - wapr, [107](#)
- lambert::lambertw, [176](#)
 - wapd, [176](#)
 - wapr, [176](#)
- lat

- onf::nf90group, 181
- latan
 - onf, 117
- latin1
 - fudu, 86
- lch_chl
 - plankton, 124
- lch_pon
 - plankton, 124
- ldlc
 - bac::dom, 166
- ldln
 - bac::dom, 166
- leap_defaults
 - julian, 105
- leap_index
 - julian, 103
- leap_table
 - julian, 106
- leaps
 - julian, 106
- lfn
 - plankton.f90, 330
- lgkspa
 - dic, 59
- lgkspc
 - dic, 59
- light
 - et::light, 177
- lnk1_sws
 - dic, 60
- lnk1_total
 - dic, 61
- lnk1p_sws
 - dic, 61
- lnk2_sws
 - dic, 62
- lnk2_total
 - dic, 62
- lnk2p_sws
 - dic, 63
- lnk3p_sws
 - dic, 63
- lnkb_total
 - dic, 64
- lnkf_free
 - dic, 65
- lnkf_total
 - dic, 65
- lnknh4_sws
 - dic, 66
- lnks_free
 - dic, 66
- lnkw_sws
 - dic, 67
- lon
 - onf::nf90group, 181
- lonan
 - onf, 117
- loss
 - det::detritus, 149
- losses
 - det::detritus, 150
- lwm1
 - lambert, 107
- maxord
 - dvode::ode, 189
- meth
 - dvode::ode, 189
- mf
 - dvode::ode, 189
- miter
 - dvode::ode, 189
- mjd_of_j2000_noon
 - julian, 106
- mort
 - cfo::zoocfo, 240
- moss
 - dvode::ode, 189
- motile
 - et::fungroup, 172
- mu0
 - cmo::phycmo, 201
- mxatol
 - dvode::ode, 190
- mxhnil
 - dvode::ode, 190
- mxrtol
 - dvode::ode, 190
- mxstep
 - dvode::ode, 190
- n2f
 - cmo::phycmo, 201
- n2p
 - bac::bacteria, 137
- name
 - et::fungroup, 172
 - onf::dimension, 162
 - onf::nf90var, 184
- names
 - dvode::ode, 190
 - et::fungroup, 172
- nbox
 - onf::nf90group, 182
- nbvds
 - onf, 118
- ncon
 - et::fungroup, 172
- nconstr
 - dvode::ode, 190
- ndims
 - onf::nf90var, 184
- neq
 - dvode::ode, 190
- nf90info

- onf, 112
- nfl
 - onf::statevars, 216
- ngdds
 - onf, 118
- ngr
 - cfo::zoocfo, 240
 - cmo::phycmo, 201
- niter
 - dic::dicsys, 158
- npfds
 - onf, 118
- nq0
 - et::fungroup, 173
- nrec
 - det::detritus, 150
- nsfds
 - onf, 118
- nsms
 - onf::statevars, 216
- nsv
 - et::fungroup, 173
- nsv_all_boxes
 - onf::statevars, 216
- nsv_one_box
 - onf::statevars, 217
- nsv_total
 - onf::statevars, 217
- nsvds
 - onf, 118
- ntim
 - onf::nf90group, 182
- nuts
 - cmo::phycmo, 201
- oc
 - et::fungroup, 173
- offsets
 - onf, 118
- omax
 - det::detritus, 150
- onf, 108
 - bfan, 117
 - dfan, 117
 - exof, 110
 - ico2ds, 117
 - infan, 117
 - innf, 110
 - inpd, 111
 - invar, 111
 - iprsds, 117
 - latan, 117
 - lonan, 117
 - nbvds, 118
 - nf90info, 112
 - ngdds, 118
 - npfds, 118
 - nsfds, 118
 - nsvds, 118
 - offsets, 118
 - onf_error, 113
 - outsv, 114
 - pdgds, 119
 - pdsds, 119
 - pfan, 119
 - plfan, 119
 - read_dim, 114
 - read_ds, 115
 - readpd, 115
 - sufan, 119
 - timan, 119
 - timunt, 120
 - write_output, 116
- onf.f90
 - set_att, 314
- onf::dataset, 109, 313
- onf::dimension, 161
 - bounds, 162
 - id, 162
 - name, 162
 - read, 161
 - units, 162
 - values, 162
 - varid, 162
- onf::nf90grd, 177
 - values, 178
- onf::nf90group, 179
 - attributes, 180
 - bounds, 180
 - depth, 180
 - depth_flux, 181
 - filename, 181
 - id, 181
 - ilat, 181
 - ilon, 181
 - info, 180
 - it0, 181
 - lat, 181
 - lon, 181
 - nbox, 182
 - ntim, 182
 - state, 182
 - time, 182
 - timediff, 182
- onf::nf90stratt, 109, 313
- onf::nf90var, 183
 - name, 184
 - ndims, 184
 - read, 183
 - type, 184
 - units, 184
 - varid, 184
- onf::nf90vec, 185
 - values, 186
- onf::phydat, 207
 - advect, 209
 - botbc, 209

- bottom, [209](#)
- cds, [209](#)
- clsbot, [209](#)
- data, [210](#)
- fpar, [210](#)
- get, [210](#)
- ibot, [210](#)
- init, [209](#)
- isflx, [210](#)
- pds, [210](#)
- profile, [210](#)
- sds, [211](#)
- surface, [211](#)
- surflux, [211](#)
- surflx, [211](#)
- surpar, [211](#)
- onf::rpd, [212](#)
- rpd, [212](#)
- onf::statevars, [213](#)
- all_states, [215](#)
- count, [215](#)
- data, [215](#)
- flux, [215](#)
- flux_depth, [216](#)
- iOflux, [216](#)
- iOsms, [216](#)
- it, [216](#)
- nfl, [216](#)
- nsms, [216](#)
- nsv_all_boxes, [216](#)
- nsv_one_box, [217](#)
- nsv_total, [217](#)
- open, [215](#)
- roc, [217](#)
- smsk, [217](#)
- soma, [217](#)
- start, [217](#)
- svgn, [217](#)
- var, [218](#)
- write_output, [218](#)
- write_soma, [218](#)
- onf_error
- onf, [113](#)
- open
- onf::statevars, [215](#)
- oppla
- oppla.F90, [325](#)
- oppla.F90
- catch_signal_2, [325](#)
- oppla, [325](#)
- outsv
- onf, [114](#)
- pachl
- cmo::phycmo, [202](#)
- par
- cmo::phycmo, [202](#)
- par_brock81
- brock81, [17](#)
- par_ld
- brock81, [18](#)
- par_meso aqua
- brock81, [18](#)
- parode
- plankton, [129](#)
- pcc
- cfo::zoocfo, [240](#)
- pdgds
- onf, [119](#)
- pds
- onf::phydat, [210](#)
- pdsds
- onf, [119](#)
- pfan
- onf, [119](#)
- ph0
- dic::dicsys, [158](#)
- phi
- cfo::zoocfo, [240](#)
- phi0
- cfo::zoocfo, [240](#)
- phim
- cfo::zoocfo, [240](#)
- phin
- cfo::zoocfo, [241](#)
- phypla
- plankton, [129](#)
- phys
- plankton, [129](#)
- pi
- et, [80](#)
- pic
- cmo::phycmo, [202](#)
- pivel
- dic::dicsys, [159](#)
- pivel_lm86
- dic, [68](#)
- pivel_w14
- dic, [68](#)
- pivel_w92
- dic, [69](#)
- pivel_w99
- dic, [70](#)
- plankton, [120](#)
- aggregate, [121](#)
- alkalinity, [122](#)
- bacpla, [128](#)
- box, [128](#)
- boxbc_cd, [122](#)
- boxbc_upwind, [123](#)
- detrit, [128](#)
- dics, [128](#)
- doma, [128](#)
- envir, [128](#)
- fpar_vertical, [123](#)
- fungrp, [129](#)
- lch_chl, [124](#)

- lch_pon, 124
- parode, 129
- phypla, 129
- phys, 129
- plankton_init, 125
- plankton_ode, 127
- states, 129
- times, 129
- zoopla, 130
- plankton.f90
 - expand_path, 330
 - lfn, 330
 - set_community, 330
 - set_environment, 330
 - set_timing, 331
- plankton::bc, 121, 329
- plankton_init
 - plankton, 125
- plankton_ode
 - plankton, 127
- plfan
 - onf, 119
- potalk
 - dic::dicsys, 159
- pp
 - cfo::zoocfo, 241
- ppm
 - cfo::zoocfo, 241
- ppn
 - cfo::zoocfo, 241
- pq
 - cfo::zoocfo, 241
- preset
 - et::preset, 212
- profile
 - onf::phydat, 210
- pt
 - dic::dicsys, 159
- pth
 - cfo::zoocfo, 241
- q0
 - et::fungroup, 173
- q0n
 - cmo::phycmo, 202
- q0p
 - cmo::phycmo, 202
- q10
 - et::fungroup, 173
- qdon
 - bac::bacteria, 138
- qdop
 - bac::bacteria, 138
- qn
 - et::fungroup, 173
- qn_param
 - bac::bacteria, 138
 - cfo::zoocfo, 242
- qp
 - et::fungroup, 174
- qp_param
 - bac::bacteria, 138
 - cfo::zoocfo, 242
- qpica
 - cmo::phycmo, 202
- qsn
 - cmo::phycmo, 203
- quiet
 - dvode::ode, 191
- quotas
 - cfo::zoocfo, 242
- r
 - cfo::zoocfo, 242
- r_tot_free
 - dic::dicsys, 159
- r_tot_sws
 - dic::dicsys, 159
- rad
 - et, 81
- ratios
 - et::fungroup, 174
- rc
 - bac::bacteria, 138
 - cmo::phycmo, 203
- rct
 - cmo::phycmo, 203
- rctht
 - cmo::phycmo, 203
- rdl
 - cmo::phycmo, 203
- rdrad
 - brock81, 18
- read
 - bac::bacteria, 135
 - bac::dom, 165
 - cfo::zoocfo, 229
 - cmo::phycmo, 196
 - det::detritus, 147
 - dic::dicsys, 154
 - dvode::ode, 187
 - et::fungroup, 170
 - onf::dimension, 161
 - onf::nf90var, 183
- read_dim
 - onf, 114
- read_ds
 - onf, 115
- read_ode
 - dvode, 71
- readpd
 - onf, 115
- relac
 - bac::dom, 166
- relan
 - bac::dom, 166
- remi
 - det::detritus, 150

- remic
 - det::detritus, [150](#)
- remichl
 - det::detritus, [150](#)
- remin
 - det::detritus, [150](#)
- remip
 - det::detritus, [151](#)
- remipic
 - det::detritus, [151](#)
- rff
 - cfo::zoocfo, [242](#)
- rffm
 - cfo::zoocfo, [242](#)
- rffn
 - cfo::zoocfo, [243](#)
- rgas
 - dic, [70](#)
- rm
 - cfo::zoocfo, [243](#)
- rm0
 - cfo::zoocfo, [243](#)
- roc
 - onf::statevars, [217](#)
- rpd
 - onf::rpd, [212](#)
- rq
 - cfo::zoocfo, [243](#)
- rrnp
 - dic::dicsys, [159](#)
- rrnsi
 - dic::dicsys, [160](#)
- rtol
 - dvode::ode, [191](#)
- schnco2
 - dic::dicsys, [160](#)
- schno2
 - dic::dicsys, [160](#)
- sdl
 - brock81, [22](#)
- sds
 - onf::phydat, [211](#)
- set
 - bac::bacteria, [135](#)
 - bac::dom, [165](#)
 - cfo::zoocfo, [229](#)
 - cmo::phycmo, [196](#)
 - det::detritus, [148](#)
 - et::fungroup, [171](#)
- set_atol
 - dvode, [72](#)
 - dvode::ode, [187](#)
- set_att
 - onf.f90, [314](#)
- set_community
 - plankton.f90, [330](#)
- set_environment
 - plankton.f90, [330](#)
- set_timing
 - plankton.f90, [331](#)
- si
 - cmo::phycmo, [203](#)
- sink
 - det::detritus, [151](#)
- sit
 - dic::dicsys, [160](#)
- slot
 - fudu, [84](#)
- smsk
 - onf::statevars, [217](#)
- so4
 - dic::dicsys, [160](#)
- solar
 - brock81, [22](#)
- soma
 - onf::statevars, [217](#)
- sparse_jacobian
 - dvode::ode, [191](#)
- start
 - onf::statevars, [217](#)
- state
 - onf::nf90group, [182](#)
- states
 - plankton, [129](#)
- stderr
 - stdunt, [130](#)
- stdin
 - stdunt, [130](#)
- stdout
 - stdunt, [130](#)
- stdunt, [130](#)
 - dp, [130](#)
 - stderr, [130](#)
 - stdin, [130](#)
 - stdout, [130](#)
- stick
 - et::fungroup, [174](#)
- sticky
 - et::fungroup, [174](#)
- sufan
 - onf, [119](#)
- sunday
 - et::sunday, [218](#)
- sunday_brock81
 - brock81, [19](#)
- sunday_const
 - brock81, [20](#)
- sunday_ld
 - brock81, [20](#)
- sunday_meso aqua
 - brock81, [21](#)
- surface
 - onf::phydat, [211](#)
- surflux
 - onf::phydat, [211](#)
- surflx

- onf::phydat, [211](#)
- surpar
 - onf::phydat, [211](#)
- svgn
 - onf::statevars, [217](#)
- svmig
 - cfo::zoocfo, [243](#)
- svph
 - cmo::phycmo, [204](#)
- switch
 - cfo::zoocfo, [243](#)
- t0ck
 - dic, [71](#)
- tch
 - cmo::phycmo, [204](#)
- tchdyn
 - cmo, [39](#)
- tchia
 - cmo, [40](#)
- thcsvm
 - cfo::zoocfo, [244](#)
- theta
 - cmo::phycmo, [204](#)
- tilt
 - brock81, [22](#)
- timan
 - onf, [119](#)
- time
 - onf::nf90group, [182](#)
- timediff
 - onf::nf90group, [182](#)
- times
 - plankton, [129](#)
- timunt
 - onf, [120](#)
- tk
 - dic::dicsys, [160](#)
- topt
 - et::fungroup, [174](#)
- totalk
 - dic::dicsys, [160](#)
- toy
 - cfo::zoocfo, [244](#)
- tref
 - et::fungroup, [174](#)
- trlc
 - bac::dom, [166](#)
- trln
 - bac::dom, [166](#)
- tspr
 - et::fungroup, [174](#)
- tvco2
 - dic::dicsys, [161](#)
- tvo2
 - dic::dicsys, [161](#)
- type
 - onf::nf90var, [184](#)
- uduerr
 - fudu, [85](#)
- units
 - et::fungroup, [175](#)
 - onf::dimension, [162](#)
 - onf::nf90var, [184](#)
- uptake
 - bac::bacteria, [138](#)
 - cmo, [40](#)
- ut_decode_time
 - fudu::ut_decode_time, [219](#)
- ut_encode_time
 - fudu::ut_encode_time, [220](#)
- ut_free
 - fudu::ut_free, [220](#)
- ut_free_system
 - fudu::ut_free_system, [221](#)
- ut_get_converter
 - fudu::ut_get_converter, [221](#)
- ut_get_status
 - fudu::ut_get_status, [222](#)
- ut_parse
 - fudu::ut_parse, [222](#)
- ut_read_xml
 - fudu::ut_read_xml, [223](#)
- ute_bad_arg
 - fudu, [86](#)
- ute_cant_format
 - fudu, [86](#)
- ute_exists
 - fudu, [86](#)
- ute_meaningless
 - fudu, [86](#)
- ute_no_second
 - fudu, [86](#)
- ute_no_unit
 - fudu, [86](#)
- ute_not_same_system
 - fudu, [86](#)
- ute_open_arg
 - fudu, [87](#)
- ute_open_default
 - fudu, [87](#)
- ute_open_env
 - fudu, [87](#)
- ute_os
 - fudu, [87](#)
- ute_parse
 - fudu, [87](#)
- ute_success
 - fudu, [87](#)
- ute_syntax
 - fudu, [87](#)
- ute_unknown
 - fudu, [87](#)
- ute_visit_error
 - fudu, [88](#)
- utencoding

- fudu, 88
- utf8
 - fudu, 88
- utsystem
 - fudu, 88
- v0
 - cmo::phycmo, 204
- v0svm
 - cfo::zoocfo, 244
- values
 - onf::dimension, 162
 - onf::nf90grd, 178
 - onf::nf90vec, 186
- var
 - onf::statevars, 218
- varid
 - onf::dimension, 162
 - onf::nf90var, 184
- vc
 - cmo::phycmo, 204
- vdic
 - cmo::phycmo, 204
- vdin
 - bac::bacteria, 138
- vdip
 - bac::bacteria, 139
- vdoc
 - bac::bacteria, 139
 - cmo::phycmo, 205
- vdom
 - bac::bacteria, 139
- vdon
 - bac::bacteria, 139
- vdop
 - bac::bacteria, 139
- vdvm
 - cfo::zoocfo, 244
- vm
 - bac::bacteria, 139
- vmax
 - bac::bacteria, 139
- vn
 - cmo::phycmo, 205
- vp
 - cmo::phycmo, 205
- vph0
 - cmo::phycmo, 205
- vpic
 - cmo::phycmo, 205
- vsink
 - cmo::phycmo, 205
- vsvm
 - cfo::zoocfo, 244
- vsvm0
 - cfo::zoocfo, 244
- vsvm1
 - cfo::zoocfo, 245
- vv
 - et::fungroup, 175
- wapd
 - lambert, 107
 - lambert::lambertw, 176
- wapr
 - lambert, 107
 - lambert::lambertw, 176
- write_output
 - onf, 116
 - onf::statevars, 218
- write_soma
 - onf::statevars, 218
- xq
 - cfo::zoocfo, 245
- yeardays
 - brock81, 23
- zc
 - cmo::phycmo, 206
- zeta
 - bac::bacteria, 139
- zn
 - cmo::phycmo, 206
- znf
 - cmo::phycmo, 206
- znu
 - cmo::phycmo, 206
- zoopla
 - plankton, 130